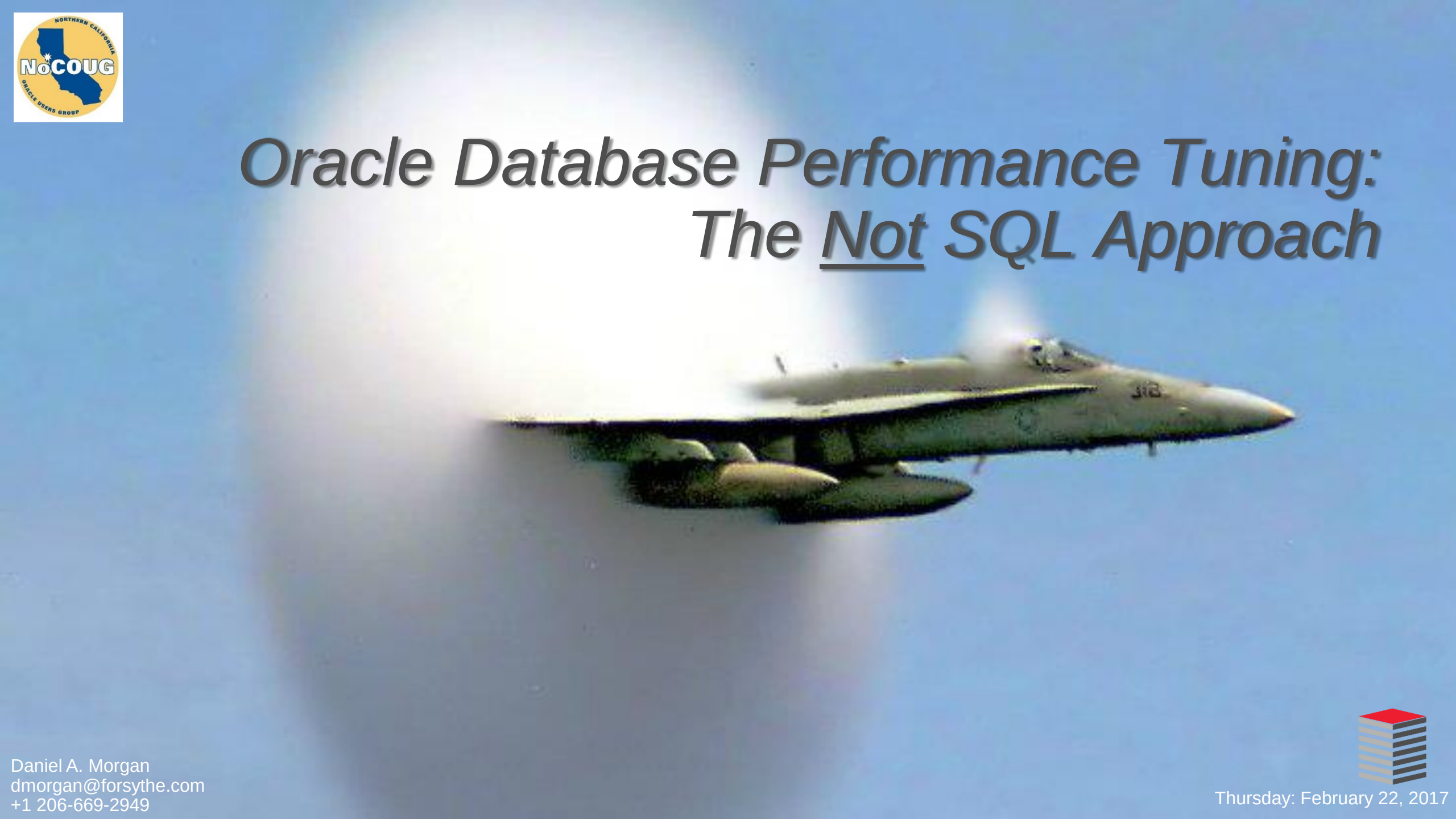
Oracle Database Performance Tuning: The Not SQL Approach



Daniel A. Morgan
dmorgan@forsythe.com
+1 206-669-2949

Thursday: February 22, 2017

Unsafe Harbor

- This room is an unsafe harbor
- You can rely on the information in this presentation to help you protect your data, your databases, your organization, and your career
- No one from Oracle has previewed this presentation
- No one from Oracle knows what I'm going to say
- No one from Oracle has supplied any of my materials
- Everything I will present is existing, proven, functionality

Introduction

Daniel Morgan

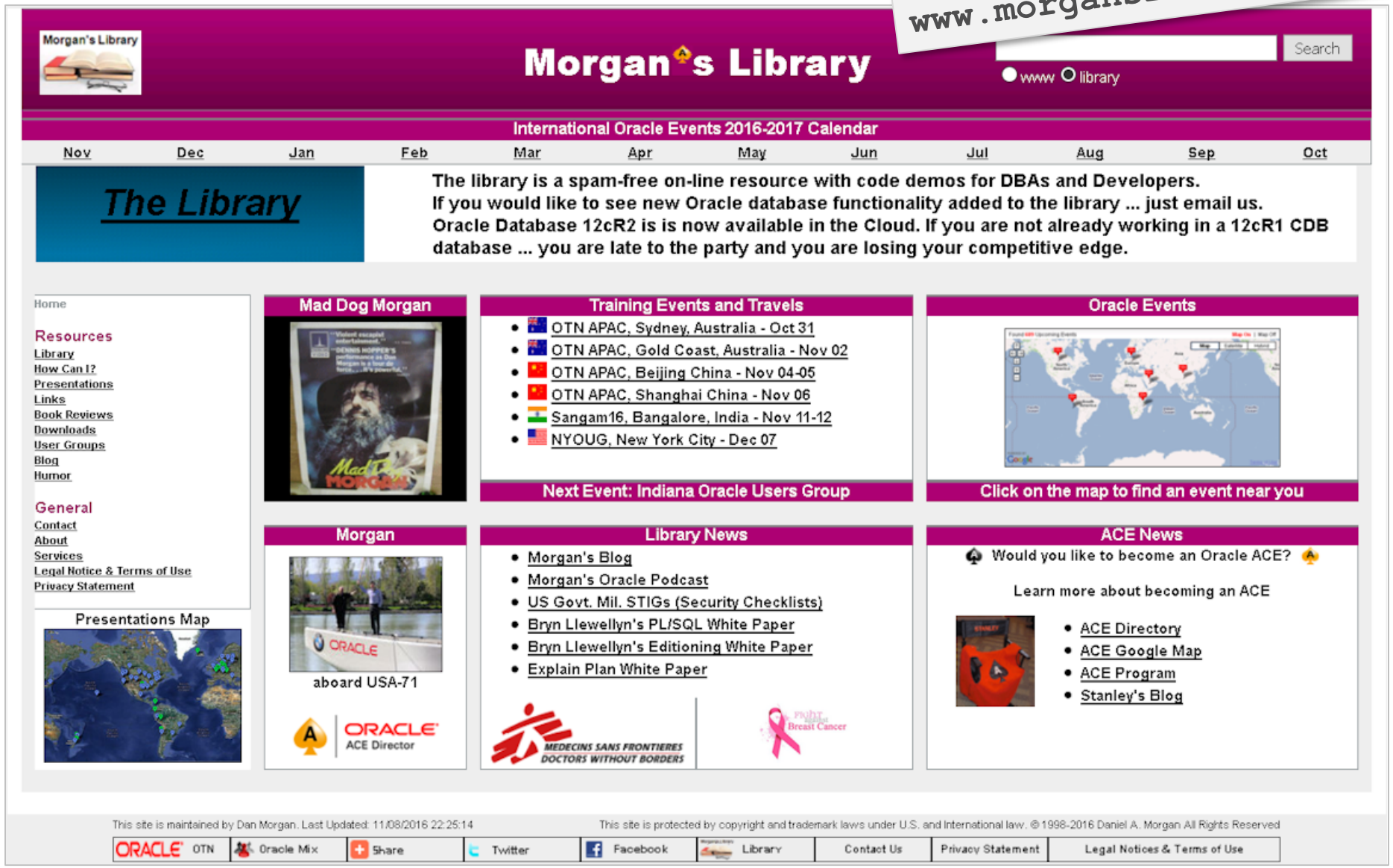
- Oracle ACE Director Alumnus
- Oracle Educator
  - Curriculum author and primary program instructor at University of Washington
  - Consultant: Harvard University
  - University Guest Lecturers
    - APAC: University of Canterbury (NZ)
    - EMEA: University of Oslo (Norway)
    - Latin America: Universidad Cenfotec, Universidad Latina de Panama, Technologico de Costa Rica
- IT Professional
  - First computer: IBM 360/40 in 1969: Fortran IV
  - Oracle Database since 1988-9 and Oracle Beta Tester
  - The Morgan behind www.morganslibrary.org
  - Member Oracle Data Integration Solutions Partner Advisory Council
  - Vice President Twin Cities Oracle Users Group (Minneapolis-St. Paul)
- Principal Adviser: Forsythe **Meta7** a Sirius Company

System/370-145 system console

email: dmorgan@forsythe.com
Twitter: @damorgan12c

My Websites: Morgan's Library

www.morganslibrary.org



www.morganslibrary.org

Meta7 In Forbes Magazine

ForbesBrandVoice® What is this?

JAN 15, 2018 @ 05:00 AM 20,020

3 Essential DBA Career Priorities For 2018

OracleVoice

Simplify IT, Drive Innovation FULL BIO

Jeff Erickson, Oracle

Many database administrators (DBAs) will go into 2018 wondering if "self-driving" databases will weaken their career prospects. More likely, 2018 will be a year that database technology leaps forward and these valuable data experts take on other, more important responsibilities.

"History is repeating itself," says longtime DBA Dan Morgan, founder of Morgan's Library and principal adviser at tech firm Meta7. Morgan has seen the DBA role evolve amid a long series of technical advances in storage, management, and performance. And each advance asked DBAs to adjust the way they work.

Meta7 In Oracle Magazine

Oracle Magazine
July – August 2017

Features
Departments
Technology & Comment Sections—
Articles and Columns

FEATURES

Great Integrations
By David Baum
Cloud-based integration reduces complexity and connects the enterprise.

Analytics for Business
By David Baum
Organizations look to the cloud to make mission-critical decisions.

Go Big, Go Metal
By Linda Currey Post
Falkonry chooses Oracle Bare Metal Cloud Services to support its pattern-recognition software.

Lessons Learned
By Jeff Erickson
Meta7 shares three top tips for moving to the cloud.

FEATURE

As Published In
ORACLE MAGAZINE
July/August 2017

Lessons Learned

By Jeff Erickson

Meta7 shares three top tips for moving to the cloud.

Meta7 knows firsthand how cloud computing is changing organizations and careers. Persistent requests from clients prompted the firm, an Oracle Platinum Partner, to purchase more than US$1.3 million worth of Oracle platform and infrastructure services to deepen its own expertise in helping customers procure and implement Oracle Cloud solutions.

Since then, the company has migrated some of its own business processes to the cloud and built many models and demos based on scenarios at clients of various sizes. "We've worked to understand everything from how a third-party on-premises application leverages Oracle Database Cloud to what's involved in a complete lift-and-shift of Oracle E-Business Suite to Oracle Cloud," says Paul Zajdel, vice president at Meta7, a division of Forsythe Technology that is dedicated to the Oracle stack.

> "All the deep-dive tuning and performance work, all the spinning up instances, the time it takes to understand how the new release handles things and explain how it's different— that's high-value, time-consuming work that DBAs don't have to do when the database is in the cloud."
>
> *– Paul Zajdel, Vice President, Meta7*

What the Meta7 team learned goes well beyond cloud service features and functions. Team members have stretched their skills with new technologies and have taken on new roles to accommodate cloud services in application architectures.

That kind of change is nothing new for Meta7 and Forsythe, which began in the early 1970s as a technology hardware leasing company. "We've reinvented ourselves several times throughout our 45-year existence," says Zajdel. It started with leasing, then reselling, then adding services, then adding security, and now adding managed services. He adds, "We're in an industry that shifts. Each time the industry shifts, we have to shift, too."

Travel Log: 2010 - Lima Peru

Travel Log: 2013 - Beijing China

Travel Log: 2014 - Galapagos Islands Ecuador

Background on Meta7

- Forsythe (Skokie, IL) is a 47 year old VAR and consultancy and the second largest Security Integrator in North America
- The Meta7 division was founded in 2015 with seed money from Oracle Corp. to focus on solving business problems for Oracle's customer
- We were purchased November 1, 2017 by Sirius Computer Solutions (San Antonio TX) and are now part of a $3.5B organization that is also the largest IBM partner on the planet

SIRIUS
The Brightest Minds in the Business

ORACLE Platinum Partner

FORSYTHE

Forbes AMERICA'S BEST MANAGEMENT CONSULTING FIRMS 2016

Forbes | 2017 AMERICA'S BEST MANAGEMENT CONSULTING FIRMS Powered by Statista

7th straight year CRN Top 50 Providers

THE CHANNEL CO. CRN SOLUTION PROVIDER 500 2017

THE CHANNEL CO. CRN TECH ELITE 250 2017

THE CHANNEL CO. CRN MSP 500 2017

Stability: IT Fire Fighting

Scalability: VLDBs and Partitioning

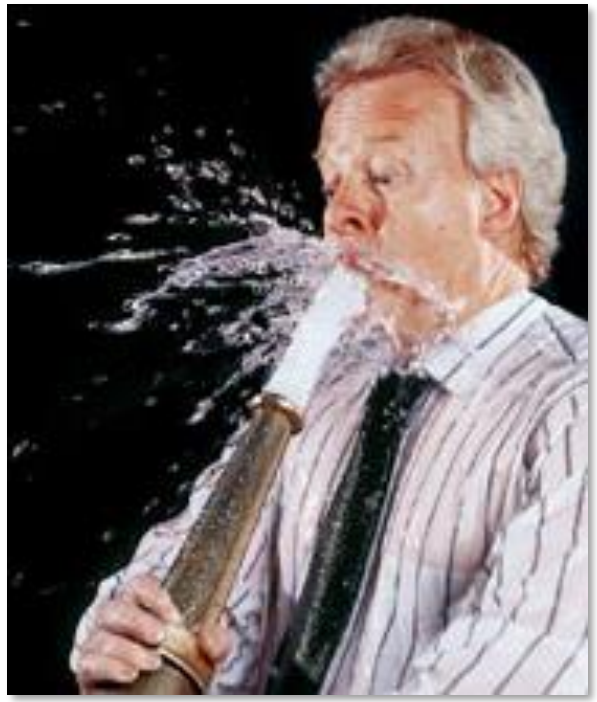
Safeguarding Databases From Cyber Threats


Zero Downtime Migration

Just In Time IT Procurement

Content Density Warning

Take Notes ... Ask Questions



Mythology versus Methodology

The #1 Database Performance Tuning Mythology

Instance Efficiency Percentages (Target 100%)

Buffer Nowait %:	99.80	Redo NoWait %:	100.00
Buffer Hit %:	97.34	In-memory Sort %:	99.99
Library Hit %:	99.97	Soft Parse %:	98.79
Execute to Parse %:	99.29	Latch Hit %:	99.57
Parse CPU to Parse Elapsd %:	0.00	% Non-Parse CPU:	96.60

This was from the worst performing Oracle Database I've seen since 2006

The #1 Database Performance Tuning Methodology

Guessing

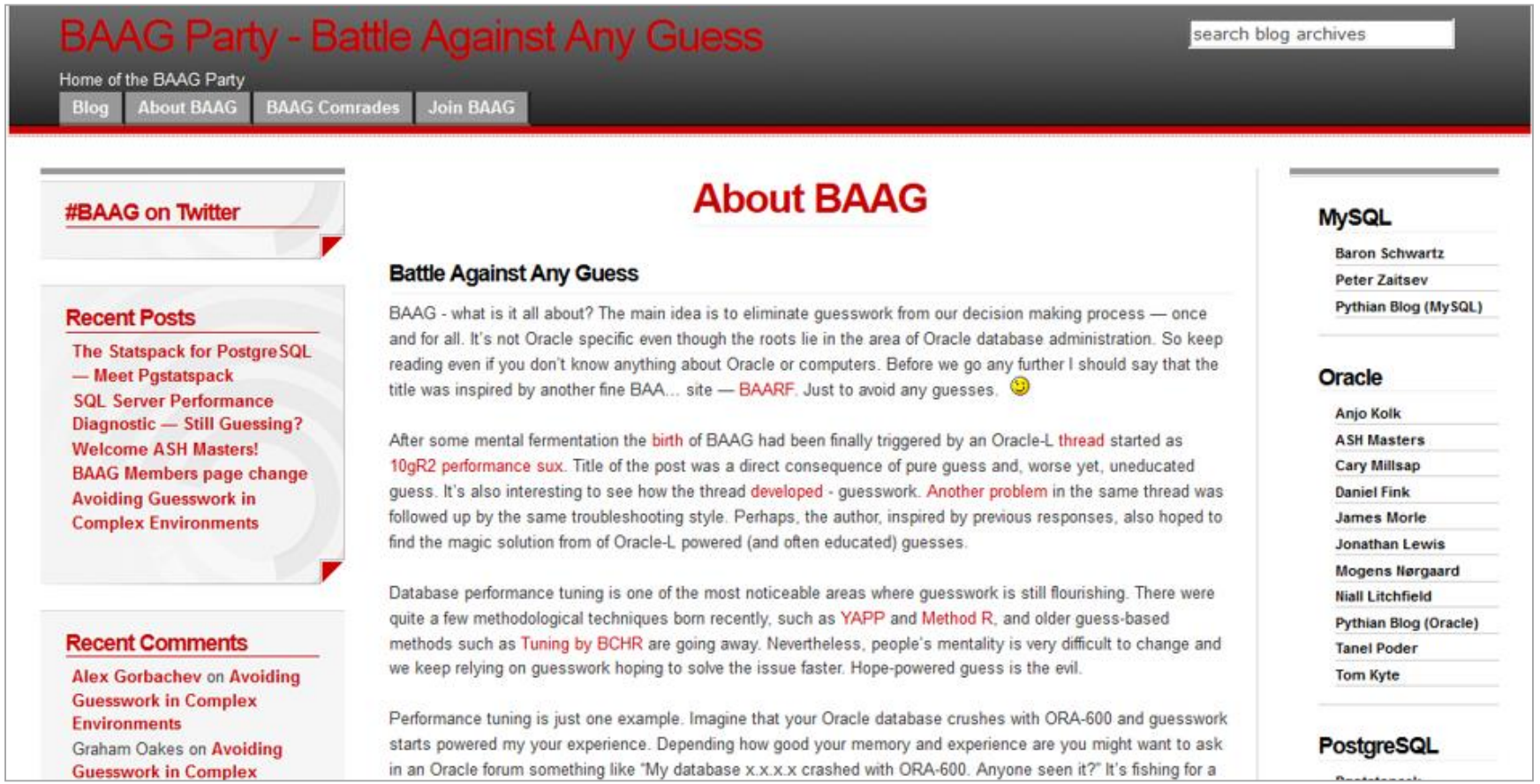
BAAG Comrades

7-Sep-08	Mohammad Illiyaz http://	BAAG Member	Because i also want to BATTLE AGAINST ANY GUESSES
22-Sep-08	Ken Jordan http://	BAAG Member	Seeking THE truth!
27-Sep-08	Martin Berger http://berx.at	BAAG Member	I travel on a sea of wild guesses, with some islands of educated guesses. Knowledge is the stars which help me traveling on these seas. I need more light!
22-Nov-08	Daniel Morgan http://www.morganslibrary.org	BAAG Member	Too often a conclusion is simply the place where someone became tired of thinking: Then the "answer" becomes a guess, a prejudice, or a retreat to mythology. I prefer the application of synapses.
11-Feb-09	Fairlie Rego http://www.el-caro.blogspot.com	BAAG Member	becoz I have had enuff
3-Mar-09	Olivier NOEL http://	BAAG Member	Help me in my daily struggle against Oracle
16-Mar-09	Michael Erwin http://www.oracle.com/consulting/technology/grid-computing.html	BAAG Member	I like sharing what I know as actual fact vs. guessing as well.
18-Mar-09	Randolf Geist http://oracle-randolf.blogspot.com/	BAAG Member	I guess (!) it's time to join.
1-Apr-09	Kevin Fries http://	BAAG Member	I guess it cannot make things worse, I've tried everything else and the Magic 8-Ball broke.
13-Apr-09	Henrik Harsfort http://	BAAG Member	to guess is to not know.
24-Apr-09	Paul Clare http://	BAAG Member	Creative problem solving is evolving into an art form.

BAAG Membership

Christian Antognini
Karl Arao
Mark Bobak
Ronald Bradford
Wolfgang Breitling
Doug Burns
Andrew Clarke
Randolf Geist
Alex Gorbachev
Marco Gralike
Frits Hoogland
John Hurley
Anjo Kolk
David Kurtz
Jonathan Lewis
Robyn Sands
Jared Still
Jeremiah Wilton

BAAG Comrades



Clearly

- No surprise ... I do not endorse guessing
- Possibly a big surprise ... I think most AWR reports are worthless
- I don't think the solutions to every performance issue is an Exadata
- And I am not attracted to all the tools with pretty GUIs
- So let us take a deep dive into performance tuning and address the root cause of the majority of issues I see in my work
- I will focus on what fixes all statements ... not just one statement
- We all know that 30+ years of doing it the way we have been ...
  - DBMS_SUPPORT introduced with version 7.2
  - DBMS_TRACE introduced with version 8.1.5
  - DBMS_MONITOR introduced with version 10.1
  - 10053 and 10046 traces with TKPROF

  ... has not eliminated performance issues

What Affects Performance (1:2)

- Hardware
  - Servers Resources
    - CPU
    - Memory
    - Bus Bandwidth and Latency
  - Storage Subsystems
  - Networks
- Software
  - Operating System Configuration
  - Virtual Machines
  - Drivers
- Database
  - Memory Allocation
  - Optimizer Configuration
  - SQL Quality
- Application
  - Web Servers
  - Application Servers
  - Middleware Caching
  - Application Code Quality

What Affects Performance (2:2)

- Everything affects performance
- What I want to focus on today is not those things that affect a **single** SQL statement
- I want to focus on those things that affect **all** SQL statements
- The overwhelming majority of database environments where I am brought in, where my team is brought in, have the following characteristics
  - The wrong servers purchased
  - Storage poorly configured
  - Networks poorly configured
  - Database installed using OUI
  - SQL originating in applications where the DBA can't fix it or written by internal developers that think the database is a byte bucket

Servers

Hardware

- Servers and Operating Systems
  - Blade Servers
  - I/O Cards
  - NUMA Architecture
  - HugePages
  - Swapiness
- Virtual Machines
- Storage
  - Controllers
  - Read-Write Caches
  - LUN Size and Layout
- Networks
  - TCP/IP
  - UDP

NUMA Memory Allocation

- Non-Uniform Memory Access
  - A memory design used in multiprocessing, where the memory access time depends on the memory location relative to the processor
  - A processor can access its own local memory faster than non-local memory
  - The benefits of NUMA are limited to particular workloads, notably on servers where the data are often associated strongly with certain tasks or users

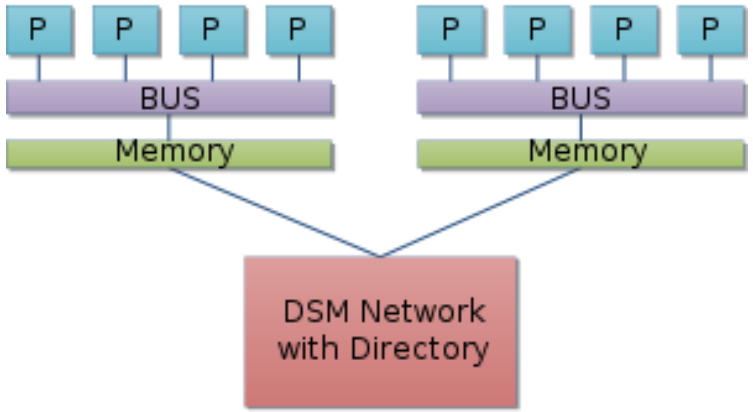


Diagram Source: Wikipedia

Detect NUMA Usage

NUMA Not Configured on an ODA

```
[root@hc1pl-oda01 etc]# numactl --hardware
available: 1 nodes (0)
node 0 size: 262086 MB
node 0 free: 113558 MB
node distances:
node   0
  0:  10

[root@hc1pl-oda01 etc]# numactl --show
policy: default
preferred node: current
physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41
42 43 44 45 46 47
cpubind: 0
nodebind: 0
membind: 0
```

NUMA Configured

```
[dmorgan@lxorap1n5 ~]$ numactl --hardware
available: 2 nodes (0-1)
node 0 size: 48457 MB
node 0 free: 269 MB
node 1 size: 48480 MB
node 1 free: 47 MB
node distances:
node   0   1
  0:  10  20
  1:  20  10

[dmorgan@lxorap1n5 ~]$ numactl --show
policy: default
preferred node: current
physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
cpubind: 0 1
nodebind: 0 1
membind: 0 1
```

more examples: www.morganslibrary.org/reference/numa.html

Is Your Database NUMA Aware?

```
SQL> SELECT a.ksppinm PNAME, c.ksppstvl PVAL, a.ksppdesc PDESC
  2  FROM x$ksppi a, x$ksppcv b, x$ksppsv c
  3  WHERE a.indx = b.indx
  4  AND a.indx = c.indx
  5  AND LOWER(a.ksppinm) LIKE '%numa%'
  6* ORDER BY 1;

PNAME                          PVAL           PDESC
------------------------------ -------------- ------------------------------------------------------
----------
_NUMA_instance_mapping         Not specified  Set of nodes that this instance should run on
_NUMA_pool_size                Not specified  aggregate size in bytes of NUMA pool
_db_block_numa                 1              Number of NUMA nodes
_enable_NUMA_interleave        TRUE           Enable NUMA interleave mode
_enable_NUMA_optimization      FALSE          Enable NUMA specific optimizations
_enable_NUMA_support           FALSE          Enable NUMA support and optimizations
_numa_buffer_cache_stats       0              Configure NUMA buffer cache stats
_numa_shift_enabled            TRUE           Enable NUMA shift
_numa_shift_value              0              user defined value for numa nodes shift
_numa_trace_level              0              numa trace event
_px_numa_stealing_enabled      TRUE           enable/disable PQ granule stealing across NUMA nodes
_px_numa_support_enabled       FALSE          enable/disable PQ NUMA support
_rm_numa_sched_enable          FALSE          Is Resource Manager (RM) related NUMA scheduled
policy enabled
_rm_numa_simulation_cpus       0              number of cpus for each pg for numa simulation in
resource mgr
_rm_numa_simulation_pgs        0              number of PGs for numa simulation in resource manager
```

more examples: www.morganslibrary.org/reference/numa.html

Enable Database NUMA Support

```
conn / as sysdba

ALTER SYSTEM SET "_enable_NUMA_support" = TRUE
COMMENT='NUMA Support Enabled 15-Mar-2015'
CONTAINER=ALL
SCOPE=SPFILE
SID='*';
```

- COMMENT
  - Commenting changes provides change management good practices
- CONTAINER
  - New 12c syntax directing a change to alter all containers or the current container
- SCOPE
  - MEMORY, SPFILE or BOTH
- SID
  - The asterisk indicates that we want the change to take place on all cluster nodes

more examples: www.morganslibrary.org/reference/numa.html

HugePages

- Also known as "Large Memory Pages" or just "Large Pages"
- Each page table entry represents a “virtual to physical” translation of a process’s memory
- Can be as large as 64 KB in size per entry
- Can be huge for large memory systems
  - See PageTables in /proc/meminfo
  - As large a 1.5 GB
- The entire SGA must fit inside the HugePages
  - If it does not fit ... then none of it will use the HugePage memory
  - You will essentially have walled your database off from using a large portion of the server's memory

Swappiness

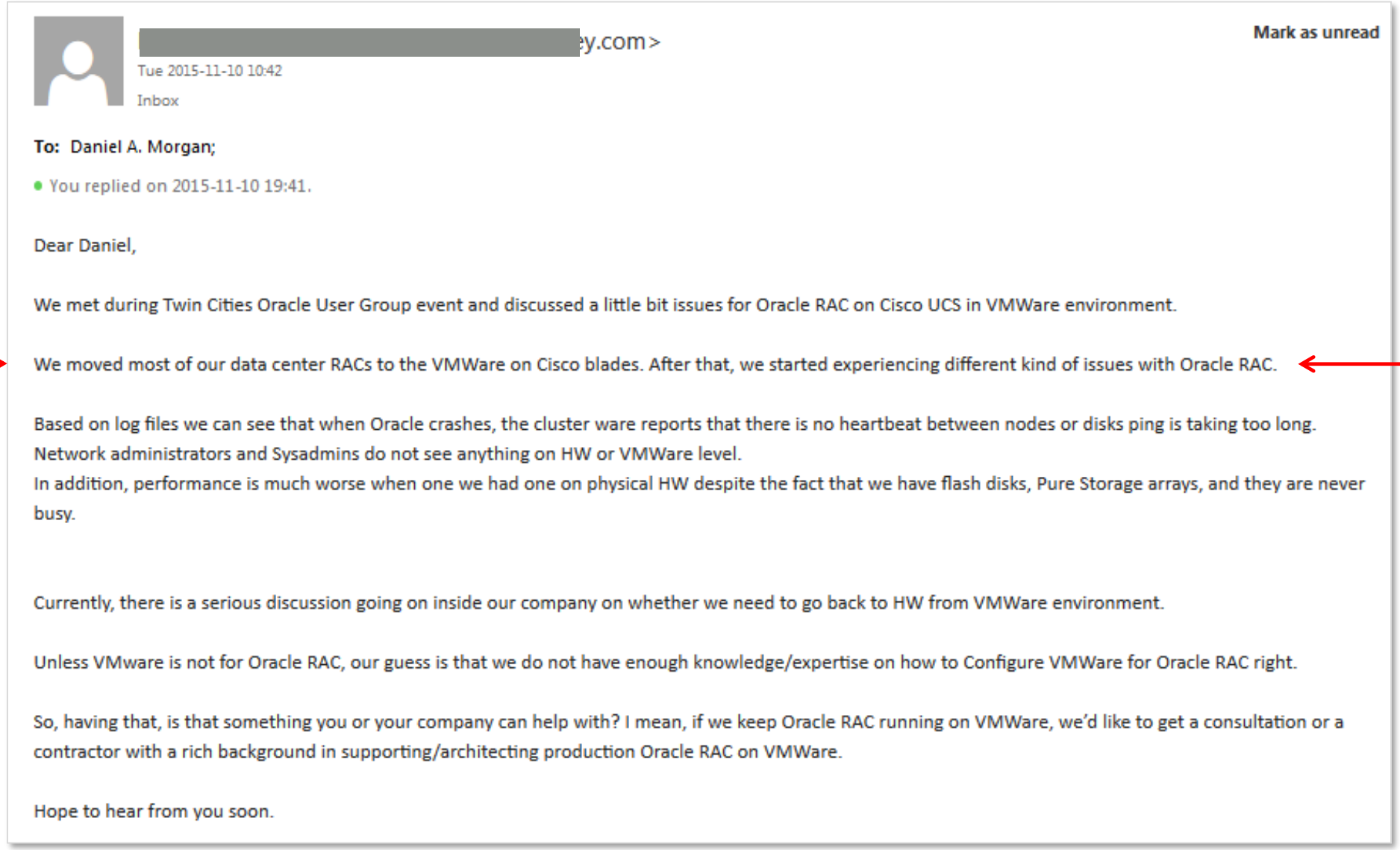
- Swapping (aka Paging)
- In a sense the operating system's version of the Oracle Temp tablespace
- Specifies a bias value for the kernel to swap out memory pages used by processes in the cgroup rather than reclaim pages from the page cache
- A value smaller than the default value of 60 reduces the kernel's preference for swapping out memory pages
- A value greater than 60 increases the preference for swapping out
- A value greater than 100 allows the system to swap out pages that fall within the address space of the cgroup's tasks

Value	Strategy
vm.swappiness=0	The kernel will swap only to avoid an out of memory condition
vm.swappiness=60	The default value
vm.swappiness=100	The kernel will swap aggressively

- In older versions of the ODA swappiness was set at 100 (a bad idea) and with the X5-2 has been set at 0 (an almost equally bad idea)

Unstable database **C**omputing **S**ystem

Tuesday Email



[redacted]ey.com>

Mark as unread

Tue 2015-11-10 10:42

Inbox

To: Daniel A. Morgan;

You replied on 2015-11-10 19:41.

Dear Daniel,

We met during Twin Cities Oracle User Group event and discussed a little bit issues for Oracle RAC on Cisco UCS in VMWare environment.

We moved most of our data center RACs to the VMWare on Cisco blades. After that, we started experiencing different kind of issues with Oracle RAC.

Based on log files we can see that when Oracle crashes, the cluster ware reports that there is no heartbeat between nodes or disks ping is taking too long.
Network administrators and Sysadmins do not see anything on HW or VMWare level.
In addition, performance is much worse when one we had one on physical HW despite the fact that we have flash disks, Pure Storage arrays, and they are never busy.

Currently, there is a serious discussion going on inside our company on whether we need to go back to HW from VMWare environment.

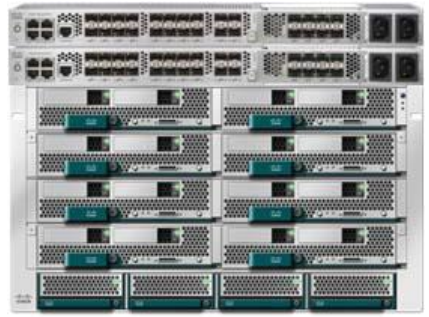
Unless VMware is not for Oracle RAC, our guess is that we do not have enough knowledge/expertise on how to Configure VMWare for Oracle RAC right.

So, having that, is that something you or your company can help with? I mean, if we keep Oracle RAC running on VMWare, we'd like to get a consultation or a contractor with a rich background in supporting/architecting production Oracle RAC on VMWare.

Hope to hear from you soon.

Let's Talk About Blades

- Stability is critical to Oracle DBAs the organizations that employ them
- If you have stability issues you can waste staggering amounts of time proving the issue isn't the database
- I have worked extensively with Cisco UCS
  - ~10 databases stand-alone 11gR2
  - ~75 RAC Active-Active and Active-Passive Failover Clusters
- The questions that need to be asked are
  - What is the value of failover for a cluster?
  - What is the value of functioning network diagnostics?
  - What is the value of stability?



VLANs and the Cluster Interconnect (1:2)

- It is essentially impossible do what is recommended in Oracle Support's "best practices" guidelines for RAC with blades: any blades from any vendor

RAC: Frequently Asked Questions (Doc ID 220970.1)

Cluster interconnect network separation can be satisfied either by using standalone, dedicated switches, which provide the highest degree of network isolation, or Virtual Local Area Networks defined on the Ethernet switch, which provide broadcast domain isolation between IP networks. VLANs are fully supported for Oracle Clusterware interconnect deployments. Partitioning the Ethernet switch with VLANs allows for:
- Sharing the same switch for private and public communication.
- Sharing the same switch for the private communication of more than one cluster.
- Sharing the same switch for private communication and shared storage access.

The following best practices should be followed:

The Cluster Interconnect VLAN must be on a non-routed IP subnet.

All Cluster Interconnect networks must be configured with non-routed IPs. The server-server communication should be single hop through the switch via the interconnect VLAN. There is no VLAN-VLAN communication.

Oracle recommends maintaining a 1:1 mapping of subnet to VLAN.

The most common VLAN deployments maintain a 1:1 mapping of subnet to VLAN. It is strongly recommended to avoid multi-subnet mapping to a single VLAN. Best practice recommends a single access VLAN port configured on the switch for the cluster interconnect VLAN. The server side network interface should have access to a single VLAN.

VLANs and the Cluster Interconnect (2:2)

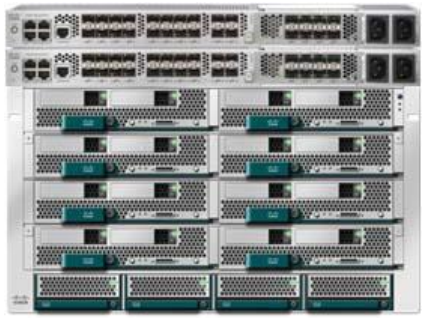
- It is extremely difficult to troubleshoot interconnect issues with UCS as the tools for separating public, storage, and fusion interconnect packets do not exist

Troubleshooting gc block lost and Poor Network Performance in a RAC Environment (Doc ID 563566.1)

6. Interconnect LAN non-dedicated
Description: Shared public IP traffic and/or shared NAS IP traffic, configured on the interconnect LAN will result in degraded application performance, network congestion and, in extreme cases, global cache block loss.

Action: The interconnect/clusterware traffic should be on a dedicated LAN defined by a non-routed subnet. Interconnect traffic should be isolated to the adjacent switch(es), e.g. interconnect traffic should not extend beyond the access layer switch(es) to which the links are attached. The interconnect traffic should not be shared with public or NAS traffic. If Virtual LANs (VLANS) are used, the interconnect should be on a single, dedicated VLAN mapped to a dedicated, non-routed subnet, which is isolated from public or NAS traffic.

My Personal Experience



- Blade servers, of which Cisco UCS is but one example, do not have sufficient independent network cards to avoid the networking becoming a single point of failure
- It is good when the public interface has a "keep alive" enabled but this is a fatal flaw for the cluster interconnect as fail-over will be delayed
- When different types of packets, public, storage, and interconnect are mixed low-level diagnostics are difficult ... if not impossible
- When different types of packets, public, storage, and interconnect are mixed the latency of one is the latency of all
- Traffic from any one blade can impact all blades

UCS Blade Server Conclusions

- Blade servers may be a good solution for application and web servers
- They may even be acceptable for stand-alone databases
- Blade servers are unsuitable when
  - High availability is the goal
  - RAC the technology for achieving it
  - Performance is critically important
  - You don't want to stay at work at night, on weekends, and holidays troubleshooting repeated unexplained failures

- **U**nstable database **C**omputing **S**ystem

Network

I/O (1:3)

- Not all HBA cards are the same
- NIC cards vary widely in capabilities and performance
  - TCP Segmentation Offloading (TSO)
  - Kernel Optimization of the TCP/IP stack

```
--enable TCP kernel auto-tuning
/proc/sys/net/ipv4/tcp_moderate_rcvbuf (1=on)

-- tune TCP max memory: tune to 2xBDP (Bandwidth x Delay Product)
-- For example, with 40 Mbits/sec bandwidth, 25 msec delay,
-- BDP = (40 x 1000 / 8 Kbytes/sec) x (0.025 sec) ~ 128 Kbytes
/proc/sys/net/ipv4/tcp_rmem and tcp_wmem 4096 87380 174760

-- tune the socket buffer sizes by setting to 2xBDP
/proc/sys/net/core/rmem_max and wmem_max

-- ensure that TCP Performance features are enabled
/proc/sys/net/ipv4/tcp_sack
/proc/sys/net/ipv4/tcp_window_scaling
/proc/sys/net/ipv4/tcp_timestamps

-- additionally be sure NIC cards have TCP off-loading capability
```

I/O (2:3)

- Optimize Data Guard

```
--sqlnet.ora
NAMES.DIRECTORY_PATH= (TNSNAMES, EZCONNECT)
DEFAULT_SDU_SIZE=32767

-- listener.ora
DGLOGSHIPB =
  (DESCRIPTION =
    (SDU = 32767)
    (SEND_BUF_SIZE=9375000)
    (RECV_BUF_SIZE=9375000)
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST =
10.0.7.2)(PORT = 1526))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = prodb)
    )
  )
```

more examples: www.morganslibrary.org/reference/data_guard.html

I/O (3:3)

- Optimize for RAC
  - Read the Oracle installation documents with very careful attention to the advice given for kernel parameters
  - Disable TCP/IP Keep-Alive for transparent failover
  - If on Linux and you don't know what rmem and wmem are ... read the docs
  - If on Solaris and you don't know what rsize and wsize are ... read the docs

Virtual Machines

- Leave sufficient cpu resources for the bare-metal operating system to perform I/O and manage network traffic
- Disable interrupt coalescing
- Disable chipset power management
- Read the docs http://www.vmware.com/pdf/Perf_Best_Practices_vSphere5.0.pdf
- VMs on NUMA machines should be configured to enhance memory allocation
  - This example is from vSphere where 0 and 1 are the processor sockets

```
numa.nodeAffinity=0,1
```

Port Exhaustion

In The Beginning (1:4)

- Customer Reports are stuck in the queue

Hi Ops

Report Jobs are getting stuck in Waiting in Queue. Also, having performance issues with Admin side

Thanks,
J

Step to Recreate

1. Log into Website
2. Navigate to Reports
3. Search for Account Data
4. Run the report for morgand
5. Notice that the report is stuck in Waiting in Queue

In The Beginning (2:4)

- The website generated an HTTP403 error

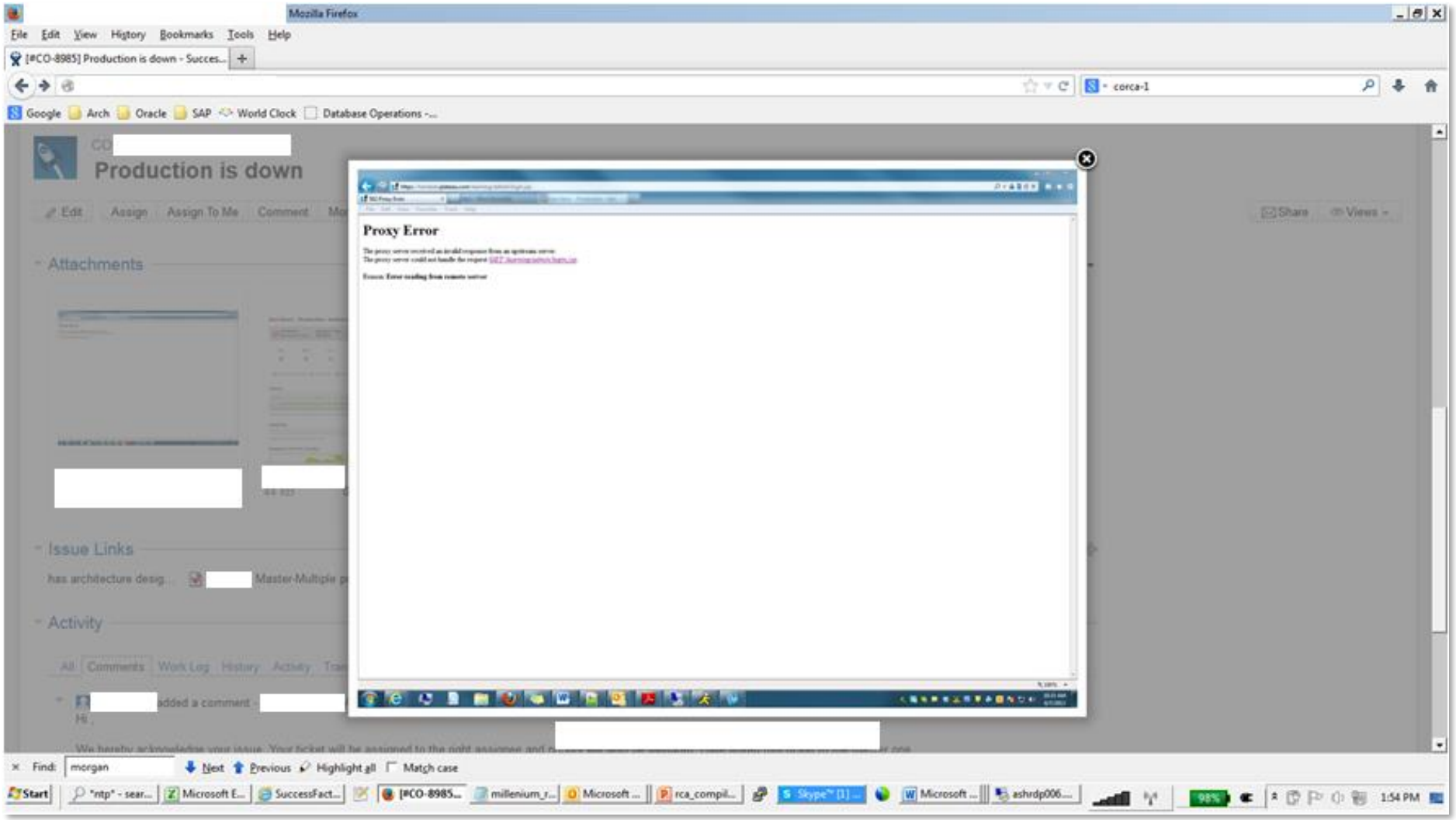
As a partner we got communication that the previously assigned sandboxes will be brought down.

Instead -as a partner- we got a them demo environment assigned (Tenant ID: PARTNER0001 which we have integrated with a customer database instance (xxxdemo ace4morgan).

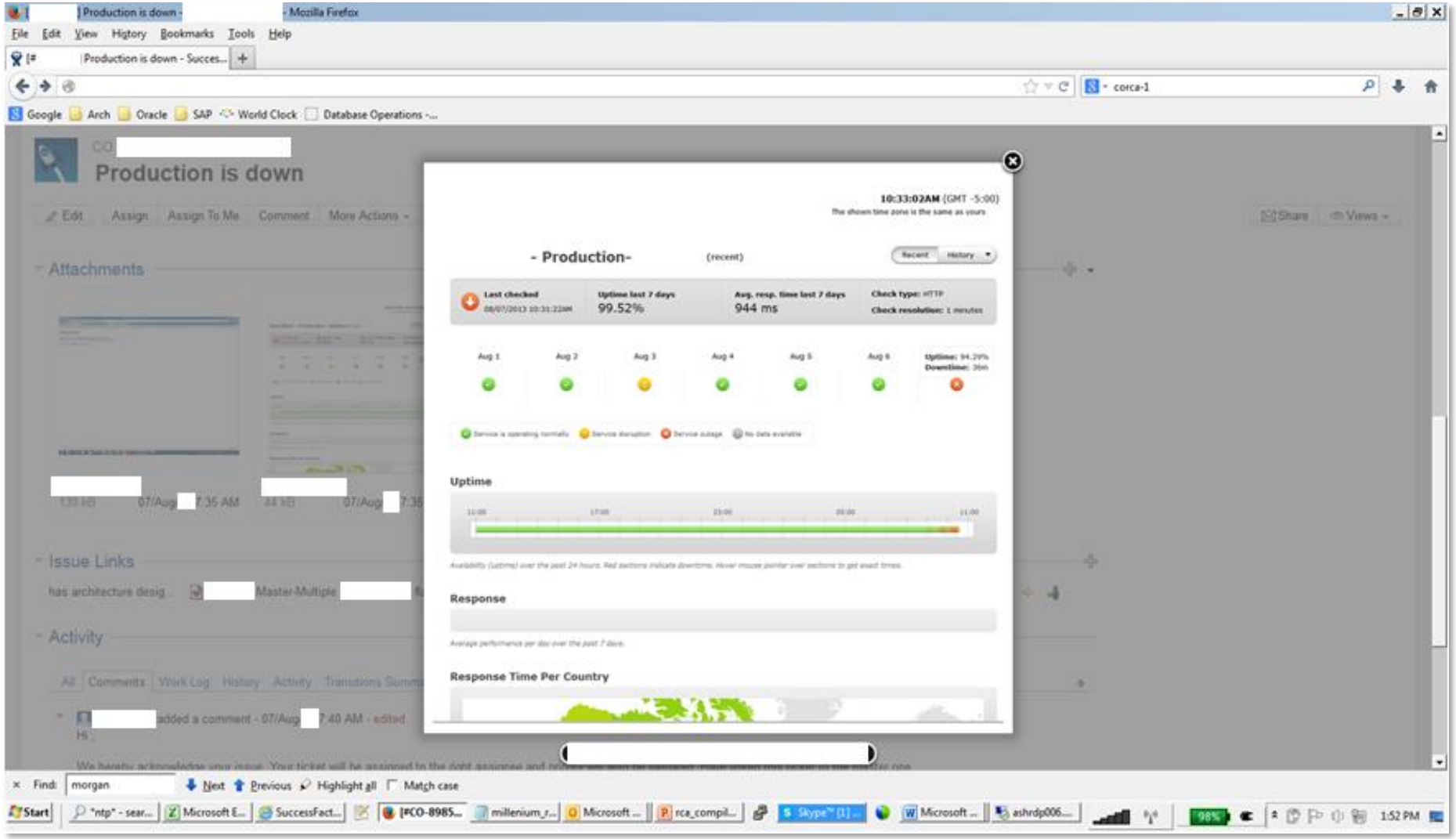
Everything was working fine (including integration).
Today I tried to access the instance via the partner and via the direct url (https://partner0001.demo.xxx.com/admin/nativelogin.jsp) but in both case an error is displayed on the screen (see attachment).

We need this be fixed as soon as possible!
(major customer demo session on Friday!)

In The Beginning (3:4)



In The Beginning (4:4)

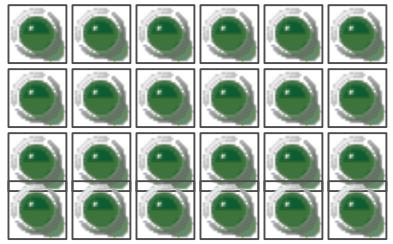


How Does An Application Server Connect to RAC?

- Do you connect to the SCAN IP by name or number?
- If a name ... a DNS server resolves the name to an IP
- To avoid single points of failure you should have two or more DNS servers with a load balancer, or two, in front of them
- The SCAN IP points to a VIP which may again need to be resolved from a name to a physical IP address
- The VIP may again point to a name which must be resolved to a physical IP address
- Most servers cache DNS entries to improve speed
  - Do you know if yours do?

Triaging a Connection Problem

- Try to connect to the cluster?
  - From where?
  - With what tool?
  - To the SCAN, VIP, or physical IP?
- Ping the IP addresses
- Run Trace Route on the IP addresses
- Read the listener log
- Read the database alert log
- Call the network admins who will tell you



everything looks good ...
the network is just Ok ...
the network is always Ok
the network will always be Ok

RESOLVE.CONF (1:3)

NAME

resolv.conf- resolver configuration file

SYNOPSIS

`/etc/resolv.conf`

DESCRIPTION

The `resolver` is a set of routines that provide access to the Internet Domain Name System. See **resolver(3RESOLV)**. `resolv.conf` is a configuration file that contains the information that is read by the `resolver` routines the first time they are invoked by a process. The file is designed to be human readable and contains a list of keywords with values that provide various types of `resolver` information.

The `resolv.conf` file contains the following configuration directives:

`nameserver`

Specifies the Internet address in dot-notation format of a name server that the resolver is to query. Up to *MAXNS* name servers may be listed, one per keyword. See `<resolv.h>`. If there are multiple servers, the resolver library queries them in the order listed. If no name server entries are present, the resolver library queries the name server on the local machine. The resolver library follows the algorithm to try a name server until the query times out. It then tries the the name servers that follow, until each query times out. It repeats all the name servers until a maximum number of retries are made.

`domain`

Specifies the local domain name. Most queries for names within this domain can use short names relative to the local domain. If no domain entry is present, the domain is determined from **sysinfo(2)** or from gethostname(3C). (Everything after the first `.' is presumed to be the domain name.) If the host name does not contain a domain part, the root domain is assumed. You can use the `LOCALDOMAIN` environment variable to override the domain name.

RESOLVE.CONF (2:3)

`search`

The search list for host name lookup. The search list is normally determined from the local domain name. By default, it contains only the local domain name. You can change the default behavior by listing the desired domain search path following the search keyword, with spaces or tabs separating the names. Most `resolver` queries will be attempted using each component of the search path in turn until a match is found. This process may be slow and will generate a lot of network traffic if the servers for the listed domains are not local. Queries will time out if no server is available for one of the domains.

The search list is currently limited to six domains and a total of 256 characters.

`sortlist` ***addresslist***

Allows addresses returned by the libresolv-internal `gethostbyname()` to be sorted. A `sortlist` is specified by IP address netmask pairs. The netmask is optional and defaults to the natural netmask of the net. The IP address and optional network pairs are separated by slashes. Up to 10 pairs may be specified. For example:

```
sortlist 130.155.160.0/255.255.240.0 130.155.0.0
```

RESOLVE.CONF (3:3)

`options`

Allows certain internal resolver variables to be modified. The syntax is

```
options option ...
```

where option is one of the following:

`debug`

Sets `RES_DEBUG` in the `_res.options` field.

`ndots:` ***n***

Sets a threshold floor for the number of dots which must appear in a name given to `res_query()` before an initial absolute (as-is) query is performed. See **resolver(3RESOLV)**. The default value for *n* is 1, which means that if there are any dots in a name, the name is tried first as an absolute name before any search list elements are appended to it.

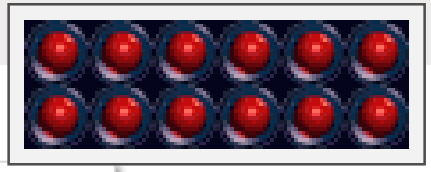
`timeout:` ***n***
`retrans:` ***n***

Sets the amount of time the resolver will wait for a response from a remote name server before retrying the query by means of a different name server. Measured in seconds, the default is `RES_TIMEOUT`. See `<resolv.h>`. The `timeout` and `retrans` values are the starting point for an exponential back off procedure where the `timeout` is doubled for every retransmit attempt.

`attempts:` ***n***
`retry:` ***n***

Sets the number of times the resolver will send a query to its name servers before giving up and returning an error to the calling application. The default is `RES_DFLRETRY`. See `<resolv.h>`.

Resolution: The DNS Admin POV (1:3)



On August 7th, we experienced a 2 hour outage that impacted more than 150 customers. In researching this outage it was noticed that DNS caching had been disabled on the Oracle Database Servers. Also, in going through the logs on the F5 Local Traffic Manager (LTM), is was noticed that there were 39,696 port exhaustion errors on port 53 going to the three DNS servers, starting at approximately 4am and ending slightly after 3pm. There were also an additional 625,665 port exhaustion error messages that were dropped in the logs, bringing the total to 665,361 port exhaustion error messages.

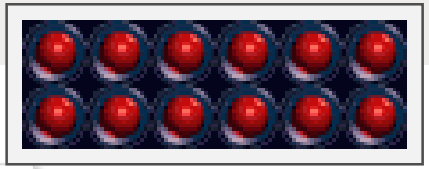
Further research discovered that there was a misconfiguration in the resolv.conf file on the servers in the data center. The resolv.conf file on these servers looked like this:

```
search morgan.priv
nameserver 10.24.244.200 (VIP pointing to servers listed below)
nameserver 10.24.244.21 (Bind server 01)
nameserver 10.24.244.25 (Bind server 02)
nameserver 10.24.244.29 (Bind server 03)
```

This results in the first DNS query going to the VIP for hostname and reverse IP resolution, and then to the three DNS servers. However, the 3 DNS servers which were supposed to be the alternative option to the VIP are also pointing to the same VIP. This basically sets up an infinite loop until the DNS queries time out.

The recommended resolution was to remove the VIP and have the servers query the DNS servers directly.

Resolution: The DNS Admin POV (2:3)



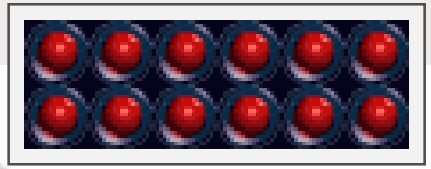
These graphs give an overview of what was happening throughout August 7th on the servers. I noticed that there is a sudden drop in connections right around 10:40am; and returning at around 10:45 am.

If you look at the files I've sent out previously, there is actually less evidence of port exhaustion between 10:22 and 10:42; with increasing levels of port exhaustion as connections and activity increases after about 12:07pm.

Additionally, I went back over the last few days and looked for port exhaustion for the DNS servers on port 53 and found the following:

```
Jul 29 -     16 port exhaustion errors
Jul 30 -      7 port exhaustion errors
Jul 31 -      8 port exhaustion errors
Aug 1  -      6 port exhaustion errors
Aug 2  - 38,711 port exhaustion errors
Aug 3  - 26,023 port exhaustion errors
Aug 4  - 22,614 port exhaustion errors
Aug 5  -     20 port exhaustion errors
Aug 6  - 11,282 port exhaustion errors
```

Resolution: The DNS Admin POV (3:3)



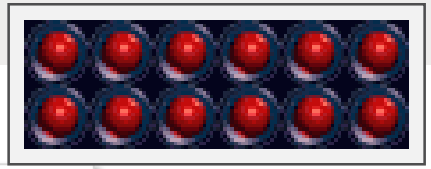
Additionally, I did some calculations on the additional port exhaustion log messages that were dropped – these were the throttling error that I mentioned previously.

On the 7th of August there were an additional 625,665 port exhaustion error messages that were dropped. On August 3rd, there were an additional 99,199 port exhaustion error messages that were dropped.

And on August 2nd, there were an additional 204,315 port exhaustion error messages that were dropped.

These numbers are in addition to the numbers of port exhaustion errors previously reported.

Resolution: The System Admin POV



Every unix box at the LAX data center has this resolv.conf file:

```
search morgan.priv
nameserver 10.24.244.200 (VIP pointing to both AD01 and AD02 windows servers)
nameserver 10.24.244.21  (Bind server 01)
nameserver 10.24.244.25  (Bind server 02)
nameserver 10.24.244.29  (Bind server 03)
```

The idea behind this design is to firstly query the VIP (for hostname resolution) and then, the 3 bind servers which are slave DNS servers of the AD DNS servers described above.

Now, I've found that the BIND servers (unix) which are supposed to be the alternative option to the VIP, have the same /etc/resolv.conf file and therefore are also pointing to the VIP on the first place. As you can imagine this basically ends up in an infinite loop until the load balancer get finally some relief or the DNS queries timeout.

Refer to the attachment "Morgan current arch" to see the workflow.

The fix should be easy and basically would consist of removing the VIP from the /etc/resolv.conf from the Bind servers and have them pointing to each AD server (bind01 -> AD01, bind02 -> AD02, etc).

The ultimate solution would be to remove the VIP from all the /etc/resolv.conf files and query the BIND servers (Helen has been asking for this for months) and although we have done that in the DEN environment, apparently that hasn't been done on the LAX side yet.

Port Exhaustion Conclusions

- As a DBA you MUST understand how DNS is configured for every one of your databases
- As a DBA you MUST understand resolv.conf and monitor it for content and changes
- As a DBA you MUST educate DNS and System Admins about how to connect to a RAC cluster or a standby
- As a DBA, when troubleshooting connection issues, you MUST log in from an application server to identify what is actually going on ... you can't just FTP to the box

Storage

Storage Hardware

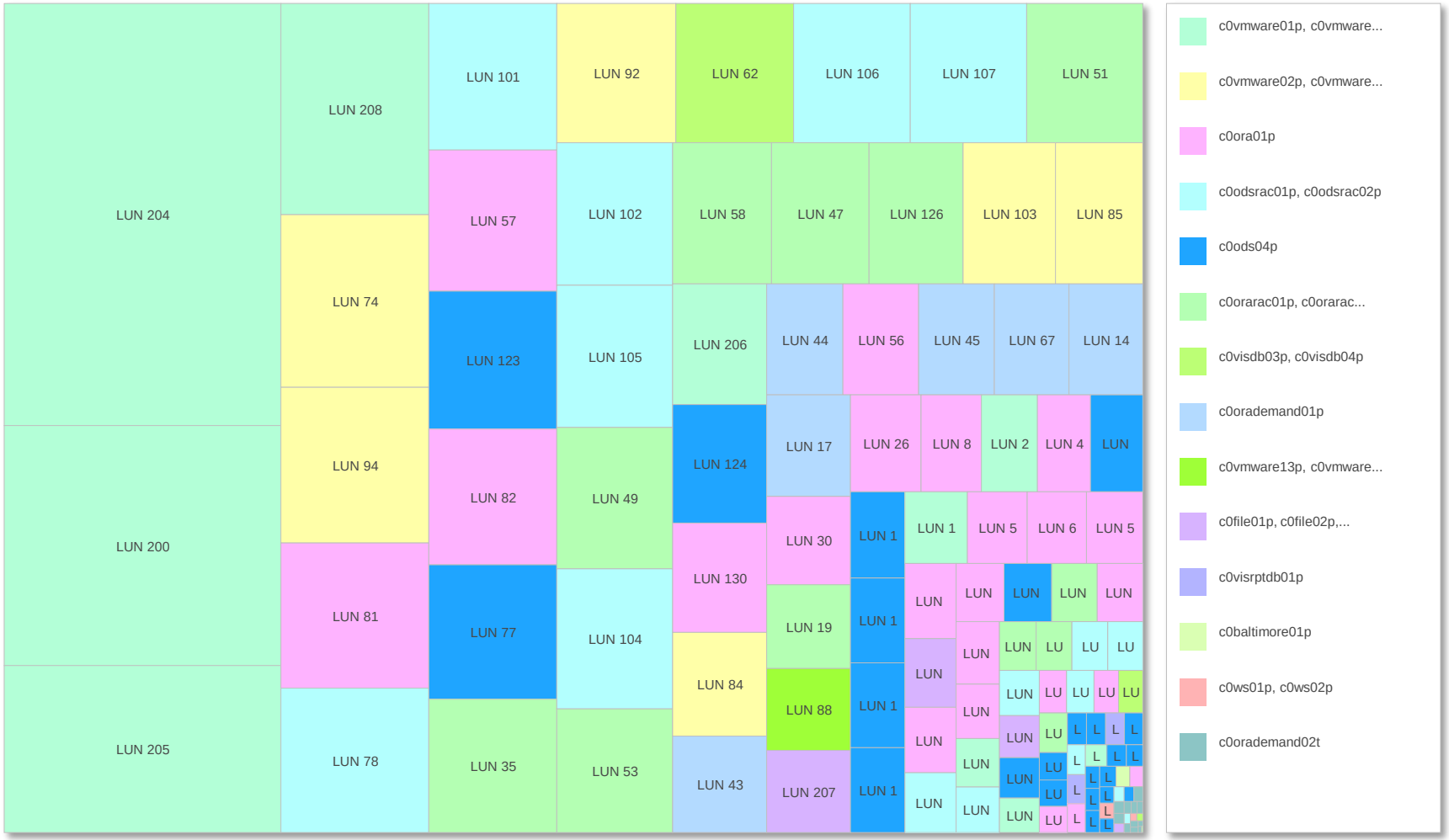
- Storage
  - Spinning Disk
  - Solid State Devices
  - Controllers
  - SAN Switches

Device	Type	IOPS	Interface	Notes
7,200 rpm SATA drives	HDD	~75-100 IOPS[2]	SATA 3 Gbit/s	
10,000 rpm SATA drives	HDD	~125-150 IOPS[2]	SATA 3 Gbit/s	
10,000 rpm SAS drives	HDD	~140 IOPS[2]	SAS	
15,000 rpm SAS drives	HDD	~175-210 IOPS[2]	SAS	

Kaminario K2	Flash/DRAM /Hybrid SSD	Up to 1,200,000 IOPS SPC-1 IOPS with the K2-D (DRAM)[41][42]	FC	
NetApp FAS6240 cluster	Flash/Disk	1,261,145 SPECsfs2008 nfsv3 IOPs using 1,440 15K disks, across 60 shelves, with virtual storage tiering.[43]	NFS, CIFS, FC, FCoE, iSCSI	SPECsfs2008 is the latest version of the Standard Performance Evaluation Corporation benchmark suite measuring file server throughput and response time, providing a standardized method for comparing performance across different vendor platforms. http://www.spec.org/sfs2008.
Fusion-io ioDrive2	SSD	Up to 9,608,000 IOPS[44]	PCIe	Only via demonstration so far.

Storage Heat Map

Storage Layout



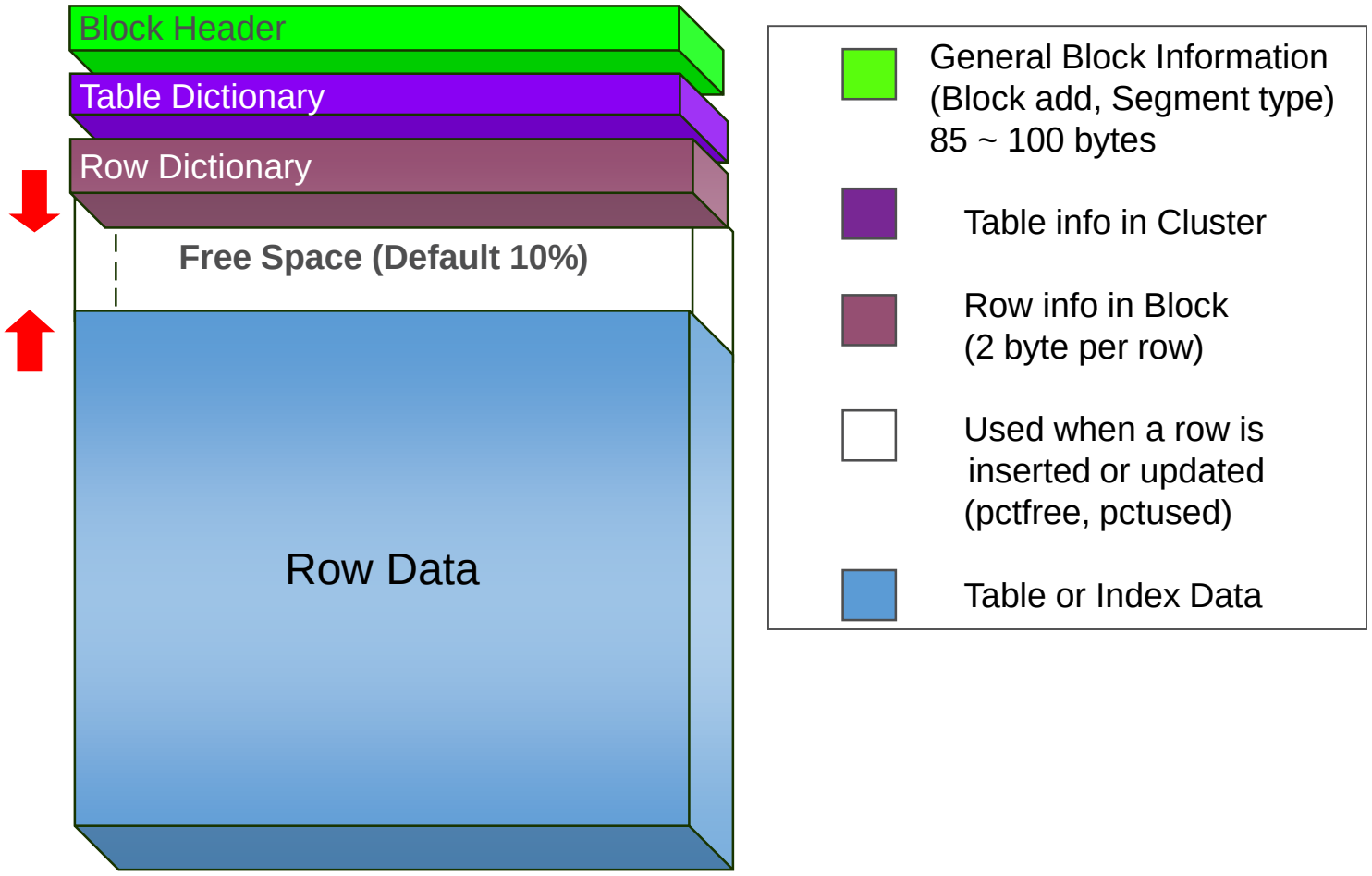
Solving the Storage Issue

- Do not used shared storage
- Do not use shared storage networks
- Do not use RAID 5
- **S**tripe **A**nd **M**irror **E**verything

In-Object Space Wastage (1:6)

- By default the Oracle Database wastes 10% of all the storage you allocate to it

```
SQL> SELECT owner, pct_free, count(*)
  2  FROM dba_tables
  3  WHERE pct_free IS NOT NULL
  3  GROUP BY owner, pct_free
  4* ORDER BY 1,2;

OWNER                PCT_FREE   COUNT(*)
-------------------- ---------- ----------
APEX_040200                   0          2
APEX_040200                  10        450
CTXSYS                        0         16
CTXSYS                       10         37
DBSNMP                        0          1
DBSNMP                       10         19
DVSYS                        10         34
GSMADMIN_INTERNAL             0          5
GSMADMIN_INTERNAL            10         14
LBACSYS                      10         22
MDSYS                        10        130
ORDDATA                      10         90
SYS                           0         90
SYS                           1         15
SYS                          10       1105
SYSTEM                        0          1
SYSTEM                       10        131
WMSYS                         0         16
WMSYS                        10         24
XDB                          10         28
XDB                          99          1
```



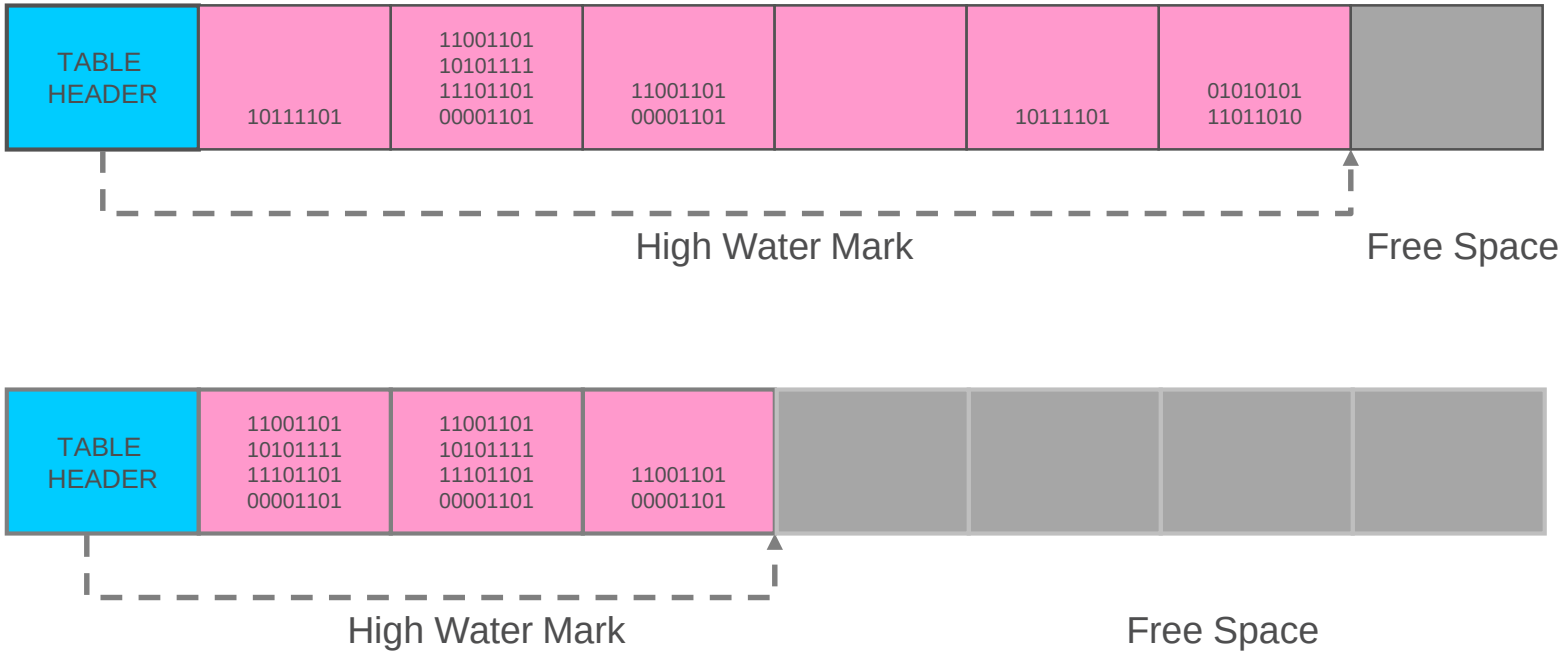
In-Object Space Wastage (2:6)

- Within file systems space may not be allocated efficiently to data files
- Within data files space may not be allocated efficiently to segments
- Within segment extents space may not be allocated efficiently too
- In the preceding example, assuming 10% free space and 90 rows per block
  - Reading 89 rows requires reading 8K
  - Reading 91 rows requires reading 16K
  - Without the pctfree loss reading 91-100 rows would still require only 8K of I/O
  - With the free space at 10% reading 200 rows requires reading 3 x 8K
  - With the free space at 0% reading 200 rows saves 1/3 of the I/O
- Know your systems well enough to know if you can eliminate the pct free value

In-Object Space Wastage (3:6)

- Online segments can be shrunk using a variety of technologies
  - DBMS_REDEFINITION
  - DBMS_SPACE
  - DDL

```
SQL> ALTER TABLE servers ENABLE ROW MOVEMENT;
SQL> ALTER TABLE servers SHRINK SPACE CASCADE;
```



Basic Compression: Heap Tables

- Works by removing duplicate values at the block level
- Sets PCT_FREE to zero

```
SQL> SELECT table_name, pct_free
  2  FROM user_tables

TABLE_NAME                       PCT_FREE
------------------------------ ----------
CAL_MONTH_SALES_MV                     10
CHANNELS                               10
COSTS
COUNTRIES                              10
CUSTOMERS                              10
DIMENSION_EXCEPTIONS                   10
DR$SUP_TEXT_IDX$I                      10
DR$SUP_TEXT_IDX$K                       0
DR$SUP_TEXT_IDX$N                       0
DR$SUP_TEXT_IDX$R                      10
FWEEK_PSCAT_SALES_MV                   10
PRODUCTS                               10
PROMOTIONS                             10
SALES
SALES_AUDIT                            10
SALES_HISTORY                          10
SALES_TRANSACTIONS_EXT                  0
SH_TEST                                 0
SUPPLEMENTARY_DEMOGRAPHICS             10
TIMES                                  10
```

```
SQL> CREATE TABLE sh_test AS
  2  SELECT /*+ APPEND */ * FROM sales;

Table created.

SQL> SELECT blocks FROM user_segments WHERE segment_name = 'SH_TEST';

    BLOCKS
----------
      4608

SQL> drop table sh_test purge;

Table dropped.

SQL> CREATE TABLE sh_test COMPRESS AS
  2  SELECT /*+ APPEND */ * FROM sales;

Table created.

SQL> SELECT blocks FROM user_segments WHERE segment_name = 'SH_TEST';

    BLOCKS
----------
      1536

 SQL> SELECT 1536/4608 FROM dual;

 1536/4608
----------
.333333333
```

Heap Table Compression Benchmark: NONE

```
SQL> CREATE TABLE sh_test AS SELECT /*+ APPEND */ * FROM sales;

Table created.

SQL> select blocks, pct_free from dba_tables where table_name = 'SH_TEST';

    BLOCKS   PCT_FREE
---------- ----------
      4513         10

SQL> explain plan for select min(time_id) from sh_test;

Explained.

SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
Plan hash value: 1514323645

------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |     1 |     8 |  1229   (1)| 00:00:01 |
|   1 |  SORT AGGREGATE    |         |     1 |     8 |            |          |
|   2 |   TABLE ACCESS FULL| SH_TEST |   918K|  7178K|  1229   (1)| 00:00:01 |
------------------------------------------------------------------------------
```

Heap Table Compression Benchmark: BASIC

```
SQL> CREATE TABLE sh_test COMPRESS AS SELECT /*+ APPEND */ * FROM sales;

Table created.

SQL> select blocks, pct_free from dba_tables where table_name = 'SH_TEST';

    BLOCKS   PCT_FREE
---------- ----------
      1511          0

SQL> explain plan for select min(time_id) from sh_test;

Explained.

SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 1514323645

--------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |     1 |     8 |   416   (2)| 00:00:01 |
|   1 |  SORT AGGREGATE    |         |     1 |     8 |            |          |
|   2 |   TABLE ACCESS FULL| SH_TEST |   918K|  7178K|   416   (2)| 00:00:01 |
--------------------------------------------------------------------------------
```

Heap Table Compression Benchmark: ADVANCED

```
SQL> CREATE TABLE sh_test COMPRESS FOR OLTP AS SELECT /*+ APPEND */ * FROM sales;

Table created.

SQL> select blocks from dba_tables where table_name = 'SH_TEST';

    BLOCKS
----------
      1676

SQL> select blocks from dba_segments where segment_name = 'SH_TEST';

    BLOCKS
----------
      1792
```

Heap Table Compression Benchmark: ADVANCED

```
SQL> CREATE TABLE sh_test COMPRESS FOR OLTP AS SELECT /*+ APPEND */ * FROM sales;

Table created.

SQL> select blocks from dba_tables where table_name = 'SH_TEST';

    BLOCKS   PCT_FREE
---------- ----------
      1676         10

SQL> CREATE TABLE sh_test ROW STORE COMPRESS ADVANCED AS SELECT /*+ APPEND */ * FROM sales;

SQL> select blocks, pct_free from dba_tables where table_name = 'SH_TEST';

    BLOCKS   PCT_FREE
---------- ----------
      1676         10

SQL> explain plan for select min(time_id) from sh_test;

Explained.

SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 1514323645

--------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |     1 |     8 |   461   (2)| 00:00:01 |
|   1 |  SORT AGGREGATE    |         |     1 |     8 |            |          |
|   2 |   TABLE ACCESS FULL| SH_TEST |   918K|  7178K|   461   (2)| 00:00:01 |
--------------------------------------------------------------------------------
```

Heap Table Compression Benchmark: Comparison

```
-------------------------------------------------------------------------------
| Id  | Operation           | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
|-----------------------------------------------------------------------------|
|   2 |   TABLE ACCESS FULL| SH_TEST |   918K|  7178K|  1229   (1)| 00:00:01 | none
|   2 |   TABLE ACCESS FULL| SH_TEST |   918K|  7178K|   416   (2)| 00:00:01 | basic
|   2 |   TABLE ACCESS FULL| SH_TEST |   918K|  7178K|   461   (2)| 00:00:01 | advanced
```

Basic Compression: B*Tree Indexes

- To compress the leading columns in an index by remove duplicate values
  - Space saving 34.4%

```
SQL> CREATE INDEX ix_sales_bic ON sales(prod_id, time_id, amount_sold);

Index created.

SQL> SELECT blocks FROM user_segments WHERE segment_name = 'IX_SALES_BIC';

    BLOCKS
----------
      4096

SQL> drop index ix_sales_bic;

Index dropped.

SQL> CREATE INDEX ix_sales_bic ON sales(prod_id, time_id, amount_sold) COMPRESS 2;

Index created.

SQL> SELECT blocks FROM user_segments WHERE segment_name = 'IX_SALES_BIC';

    BLOCKS
----------
      2688
```

Advanced Compression: B*Tree Indexes

- New feature in Database 12cR1
  - Specify COMPRESS ADVANCED LOW to enable advanced index compression. Advanced index compression improves compression ratios significantly while still providing efficient access to indexes. Therefore, advanced index compression works well on all supported indexes, including those indexes that are not good candidates for prefix compression.
  - Space saving 20%

```
SQL> CREATE INDEX index ix_sales_amnt_sold ON sales(amount_sold);

Index created.

SQL> SELECT blocks FROM user_segments WHERE segment_name = 'IX_SALES_AMNT_SOLD';

    BLOCKS
----------
      2560

SQL> drop index ix_sales_amnt_sold;

Index dropped.

SQL> CREATE INDEX ix_sales_amnt_sold ON sales(amount_sold) COMPRESS ADVANCED LOW;

Index created.

SQL> SELECT blocks FROM user_segments WHERE segment_name = 'IX_SALES_AMNT_SOLD';

    BLOCKS
----------
      2048
```

DBMS_COMPRESSION Built-in Package

- Supports advanced compression features new as of 11gR2 and the Oracle-Sun Exadata Server
- Stored procedures include
  - DUMP_COMPRESSION_MAP
  - GET_COMPRESSION_RATIO
  - GET_COMPRESSION_TYPE
  - INCREMENTAL_COMPRESS

```
SQL> DECLARE
  2  blkcnt_cmp   PLS_INTEGER;
  3  blkcnt_uncmp PLS_INTEGER;
  4  row_comp     PLS_INTEGER;
  5  row_uncmp    PLS_INTEGER;
  6  cmp_ratio    NUMBER;
  7  comptype     VARCHAR2(30);
  8  BEGIN
  9    dbms_compression.get_compression_ratio('SYSTEM', 'SYS',
'SOURCE$', NULL, dbms_compression.comp_advanced, blkcnt_cmp,
blkcnt_uncmp, row_comp, row_uncmp, cmp_ratio, comptype);
 10    dbms_output.put_line('Block Count Compressed:   ' || TO_CHAR(blkcnt_cmp));
 11    dbms_output.put_line('Block Count UnCompressed: ' || TO_CHAR(blkcnt_uncmp));
 12    dbms_output.put_line('Row Count Compressed:     ' || TO_CHAR(row_comp));
 13    dbms_output.put_line('Row Count UnCompressed:   ' || TO_CHAR(row_uncmp));
 14    dbms_output.put_line('Block Count Compressed:   ' || TO_CHAR(cmp_ratio));
 15    dbms_output.put_line('Compression Type:         ' || comptype);
 16* END;
 17  /
Block Count Compressed:   1749
Block Count UnCompressed: 1894
Row Count Compressed:     58
Row Count UnCompressed:   53
Block Count Compressed:   1
Compression Type:         "Compress Advanced"

PL/SQL procedure successfully completed.
```

In-Object Space Wastage (4:6)

- DBMS_SPACE Built-In Package
  - Fully documented and supported
  - FREE_BLOCKS
  - SPACE_USAGE
  - UNUSED_SPACE
  - VERIFY_SHRINK_CANDIDATE
  - VERIFY_SHRINK_CANDIDATE_TBF

In-Object Space Wastage (5:6)

- No wastage

```
SQL> DECLARE
  2  uf   NUMBER;
  3  ub   NUMBER;
  4  f1   NUMBER;
  5  f1b  NUMBER;
  6  f2   NUMBER;
  7  f2b  NUMBER;
  8  f3   NUMBER;
  9  f3b  NUMBER;
 10  f4   NUMBER;
 11  f4b  NUMBER;
 12  fbl  NUMBER;
 13  fby  NUMBER;
 14  BEGIN
 15    dbms_space.space_usage('UWCLASS','SERVERS', 'TABLE', uf, ub, f1, f1b, f2, f2b, f3, f3b, f4, f4b, fbl, fby);
 16
 17    dbms_output.put_line('unformatted blocks:  ' || TO_CHAR(uf));
 18    dbms_output.put_line('unformatted bytes:   ' || TO_CHAR(ub));
 19    dbms_output.put_line('blocks 0-25% free:   ' || TO_CHAR(f1));
 20    dbms_output.put_line('bytes 0-25% free:    ' || TO_CHAR(f1b));
 21    dbms_output.put_line('blocks 25-50% free:  ' || TO_CHAR(f2));
 22    dbms_output.put_line('bytes 25-50% free:   ' || TO_CHAR(f2b));
 23    dbms_output.put_line('blocks 50-75% free:  ' || TO_CHAR(f3));
 24    dbms_output.put_line('bytes 50-75% free:   ' || TO_CHAR(f3b));
 25    dbms_output.put_line('blocks 75-100% free: ' || TO_CHAR(f4));
 26    dbms_output.put_line('bytes 75-100% free:  ' || TO_CHAR(f4b));
 27    dbms_output.put_line('full blocks:         ' || TO_CHAR(fbl));
 28    dbms_output.put_line('full bytes:          ' || TO_CHAR(fby));
 29  END;
 30  /
unformatted blocks:  0
unformatted bytes:   0
blocks 0-25% free:   0
bytes 0-25% free:    0
blocks 25-50% free:  0
bytes 25-50% free:   0
blocks 50-75% free:  0
bytes 50-75% free:   0
blocks 75-100% free: 0
bytes 75-100% free:  0
full blocks:         2
full bytes:          16384
```

In-Object Space Wastage (6:6)

- Minor wastage

```
SQL> DECLARE
  2  uf   NUMBER;
  3  ub   NUMBER;
  4  f1   NUMBER;
  5  f1b  NUMBER;
  6  f2   NUMBER;
  7  f2b  NUMBER;
  8  f3   NUMBER;
  9  f3b  NUMBER;
 10  f4   NUMBER;
 11  f4b  NUMBER;
 12  fbl  NUMBER;
 13  fby  NUMBER;
 14  BEGIN
 15    dbms_space.space_usage('XDB','X$QN40ORNNWS4T9IVANV2GK293AFHS', 'TABLE', uf, ub, f1, f1b, f2, f2b, f3, f3b, f4, f4b, fbl, fby);
 16
 17    dbms_output.put_line('unformatted blocks:  ' || TO_CHAR(uf));
 18    dbms_output.put_line('unformatted bytes:   ' || TO_CHAR(ub));
 19    dbms_output.put_line('blocks 0-25% free:   ' || TO_CHAR(f1));
 20    dbms_output.put_line('bytes 0-25% free:    ' || TO_CHAR(f1b));
 21    dbms_output.put_line('blocks 25-50% free:  ' || TO_CHAR(f2));
 22    dbms_output.put_line('bytes 25-50% free:   ' || TO_CHAR(f2b));
 23    dbms_output.put_line('blocks 50-75% free:  ' || TO_CHAR(f3));
 24    dbms_output.put_line('bytes 50-75% free:   ' || TO_CHAR(f3b));
 25    dbms_output.put_line('blocks 75-100% free: ' || TO_CHAR(f4));
 26    dbms_output.put_line('bytes 75-100% free:  ' || TO_CHAR(f4b));
 27    dbms_output.put_line('full blocks:         ' || TO_CHAR(fbl));
 28    dbms_output.put_line('full bytes:          ' || TO_CHAR(fby));
 29  END;
 30  /
unformatted blocks:  0
unformatted bytes:   0
blocks 0-25% free:   0
bytes 0-25% free:    0
blocks 25-50% free:  0
bytes 25-50% free:   0
blocks 50-75% free:  1
bytes 50-75% free:   8192
blocks 75-100% free: 2
bytes 75-100% free:  16384
full blocks:         2
full bytes:          16384
```

Other Storage Optimizations

- Oracle's default tablespace settings waste disk and create unnecessary I/O in the vast majority of databases
- Default smallfile tablespaces should not be used in environments where the number of datafiles, in a tablespace, increases over time
- Default table creations are almost always inefficient
  - Heap tables are the default choice not necessarily the best choice
- Oracle optimizes data dictionary performance by clustering tables ... do you?
- New in 12c ... Attribute clustering specifies how to cluster data in close physical proximity based on column contents which improves compression, indexes, and zone maps

```
CREATE TABLE cluster_t (
 rid        NUMBER,
 lname      VARCHAR2(25),
 state_prov VARCHAR2(2))
 CLUSTERING BY LINEAR ORDER (state_prov) WITH MATERIALIZED ZONEMAP;
```

One Issue with Attribute Clustering

- Oops!

```
SQL> drop table cluster_t purge;

Table dropped.

SQL> CREATE TABLE cluster_t (
  2  rid        NUMBER,
  3  lname      VARCHAR2(25),
  4  state_prov VARCHAR2(2))
  5  CLUSTERING BY LINEAR ORDER (state_prov) WITH MATERIALIZED ZONEMAP;
CREATE TABLE cluster_t (
*
ERROR at line 1:
ORA-01031: insufficient privileges

SQL> desc cluster_t
 Name                      Null?    Type
 ------------------------- -------- -------------
 RID                       NUMBER
 LNAME                     VARCHAR2(25)
 STATE_PROV                VARCHAR2(2)
```

- Opening an SR

SQL

I'm Not Afraid To Show You Mine

Page One

0cjpvrvqy854kh	SELECT 'ERICSSON_CSCORE.EOS' AS TABLENAME, sunday, CASE WHEN percent(summarised_ontime_count, (ontime_count-late_count), 2) > 100 THEN 100 ELSE percent(summarised_ontime_count, (ontime_count-late_count), 2) END percent_of_cutoff, case when 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) < 0 then 0 else 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) end percent_of_cutoff_not_summed, percent(summarised_ontime_count, ontime_count+late_count, 2)percentage_of_total_by5am, percent(summarised_ontime_count+summarised_late_count, ontime_count+late_count, 2) current_percentage, ontime_count+late_count current_raw_elements, summarised_ontime_count+summarised_late_count current_summary_elements, summarised_ontime_count-summarised_late_count SUM_ELEMENTS_AT5AM, late_count late_raw_elements, ontime_count-late_count available_at_cutoff, percent(num_of_sumrows, num_of_rows, 2) accuracy from (select count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+3/24 then MSC end) late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+3/24 then MSC end) ontime_count, count (*) num_of_rows from ERICSSON_CSCORE.EOS where datetime between trunc(sysdate)-1 and trunc(sysdate)-1/24/60)rawt, (select max(datetime)sunday, count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+7/24 then MSC end) summarised_late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(s ysdate)+7/24 then MSC end) summarised_ontime_count, sum(entries) num_of_sumrows from ERICSSON_CSCORE.EOS_DY where datetime = trunc(sysdate)-1)sumt union all SELECT 'ERICSSON_CSCORE.MSCSTATS' AS TABLENAME, sunday, CASE WHEN percent(summarised_ontime_count, (ontime_count-late_count), 2) > 100 THEN 100 ELSE percent(summarised_ontime_count, (ontime_count-late_count), 2) END percent_of_cutoff, case when 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) < 0 then 0 else 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) end percent_of_cutoff_not_summed, percent(summarised_ontime_count, ontime_count+late_count, 2)percentage_of_total_by5am, percent(summarised_ontime_count+summarised_late_count, ontime_count+late_count, 2) current_percentage, ontime_count+late_count current_raw_elements, summarised_ontime_count+summarised_late_count current_summary_elements, summarised_ontime_count-summarised_late_count SUM_ELEMENTS_AT5AM, late_count late_raw_elements, ontime_count-late_count available_at_cutoff, percent(num_of_sumrows, num_of_rows, 2) accuracy from (select count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+3/24 then MSC end) late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+3/24 then MSC end) ontime_count, count (*) num_of_rows from ERICSSON_CSCORE.MSCSTATS where datetime between trunc(sysdate)-1 and trunc(sysdate)-1/24/60)rawt, (select max(datetime)sunday, count(dist inct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+7/24 then MSC end) summarised_late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+7/24 then MSC end) summarised_ontime_count, sum(entries) num_of_sumrows from ERICSSON_CSCORE.MSCSTATS_DY where datetime = trunc(sysdate)-1)sumt union all SELECT 'ERICSSON_CSCORE.MTRAFTYPE' AS TABLENAME, sunday, CASE WHEN percent(summarised_ontime_count, (ontime_count-late_count), 2) > 100 THEN 100 ELSE percent(summarised_ontime_count, (ontime_count-late_count), 2) END percent_of_cutoff, case when 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) < 0 then 0 else 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) end percent_of_cutoff_not_summed, percent(summarised_ontime_count, ontime_count+late_count, 2)percentage_of_total_by5am, percent(summarised_ontime_count+summarised_late_count, ontime_count+late_count, 2) current_percentage, ontime_count+late_count current_raw_elements, summarised_ontime_count+summarised_late_count current_summary_elements, summarised_ontime_count-summarised_late_count SUM_ELEMENTS_AT5AM, late_count late_raw_elements, ontime_count-late_count available_at_cutoff, percent(num_of_sumrows, num_of_rows, 2) accuracy from (select count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+3/24 then MSC end) late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+3/24 then MSC end) ontime_count , count (*) num_of_rows from ERICSSON_CSCORE.MTRAFTYPE where datetime between trunc(sysdate)-1 and trunc(sysdate)-1/24/60)rawt, (select max(datetime)sunday, count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+7/24 then MSC end) summarised_late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+7/24 then MSC end) summarised_ontime_count, sum(entries) num_of_sumrows from ERICSSON_CSCORE.MTRAFTYPE_DY where datetime = trunc(sysdate)-1)sumt union all SELECT 'ERICSSON_GERAN.CELLGPRS' AS TABLENAME, sunday, CASE WHEN percent(summarised_ontime_count, (ontime_count-late_count), 2) > 100 THEN 100 ELSE percent(summarised_ontime_count, (ontime_count-late_count), 2) END percent_of_cutoff, case when 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) < 0 then 0 else 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) end percent_of_cutoff_not_summed, percent(summarised_ontime_count, ontime_count+late_count, 2)percentage_of_total_by5am, percent(summarised_ontime_count+summarised_late_count, ontime_count+late_count, 2) current_percentage, ontime_count+late_count current_raw_elements, summarised_ontime_count+summarised_late_count current_summary_elements, summarised_ontime_count-summarised_late_count SUM_ELEMENTS_AT5AM, late_count late_raw_elements, ontime_count-late_count available_at_cutoff, percent(num_of_sumrows, num_of_rows, 2) accuracy from (select count(distinct case when trunc(d atetime_ins, 'mi') > trunc(sysdate)+3/24 then BSC end) late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+3/24 then BSC end) ontime_count, count (*) num_of_rows from ERICSSON_GERAN.CELLGPRS where datetime between trunc(sysdate)-1 and trunc(sysdate)-1/24/60)rawt, (select max(datetime)sunday, count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+7/24 then BSC end) summarised_late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+7/24 then BSC end) summarised_ontime_count, sum(entries) num_of_sumrows from ERICSSON_GERAN.CELLGPRS_DY where datetime = trunc(sysdate)-1)sumt union all SELECT 'ERICSSON_GERAN.CELLSTATS' AS TABLENAME, sunday, CASE WHEN percent(summarised_ontime_count, (ontime_count-late_count), 2) > 100 THEN 100 ELSE percent(summarised_ontime_count, (ontime_count-late_count), 2) END percent_of_cutoff, case when 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) < 0 then 0 else 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) end percent_of_cutoff_not_summed, percent(summarised_ontime_count, ontime_count+late_count, 2)percentage_of_total_by5am, percent(summarised_ontime_count+summarised_late_count, ontime_count+late_count, 2) current_percentage, ontime_count+late_count current_raw_elements, summarised_ontime_count+summarised_late_count current_summary_elements, summarised_ontime_count-summarised_late_count SUM_ELEMENTS_AT5AM, late_ count late_raw_elements, ontime_count-late_count available_at_cutoff, percent(num_of_sumrows, num_of_rows, 2) accuracy from (select count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+3/24 then BSC end) late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+3/24 then BSC end) ontime_count, count (*) num_of_rows from ERICSSON_GERAN.CELLSTATS where datetime between trunc(sysdate)-1 and trunc(sysdate)-1/24/60)rawt, (select max(datetime)sunday, count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+7/24 then BSC end) summarised_late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+7/24 then BSC end) summarised_ontime_count, sum(entries) num_of_sumrows from ERICSSON_GERAN.CELLSTATS_DY where datetime = trunc(sysdate)-1)sumt union all SELECT 'ERICSSON_PSCORE.MM' AS TABLENAME, sunday, CASE WHEN percent(summarised_ontime_count, (ontime_count-late_count), 2) > 100 THEN 100 ELSE percent(summarised_ontime_count, (ontime_count-late_count), 2) END percent_of_cutoff, case when 100-percent(summarised_ontime_count,

Page Two

(ontime_count-late_count), 2) < 0 then 0 else 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) end percent_of_cutoff_not_summed, percent(summarised_ontime_count, ontime_count+late_count, 2)percentage_of_total_by5am, percent(summarised_ontime_count+summarised_late_count, ontime_count+late_count, 2) current_percentage, ontime_count+late_count curr ent_raw_elements, summarised_ontime_count+summarised_late_count current_summary_elements, summarised_ontime_count-summarised_late_count SUM_ELEMENTS_AT5AM, late_count late_raw_elements, ontime_count-late_count available_at_cutoff, percent(num_of_sumrows, num_of_rows, 2) accuracy from (select count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+3/24 then SGSNID end) late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+3/24 then SGSNID end) ontime_count, count (*) num_of_rows from ERICSSON_PSCORE.MM where datetime between trunc(sysdate)-1 and trunc(sysdate)-1/24/60)rawt, (select max(datetime)sumday, count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+7/24 then SGSNID end) summarised_late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+7/24 then SGSNID end) summarised_ontime_count, sum(entries) num_of_sumrows from ERICSSON_PSCORE.MM_DY where datetime = trunc(sysdate)-1)sumt union all SELECT 'ERICSSON_PSCORE.SGSNSTAT' AS TABLENAME, sumday, CASE WHEN percent(summarised_ontime_count, (ontime_count-late_count), 2) > 100 THEN 100 ELSE percent(summarised_ontime_count, (ontime_count-late_count), 2) END percent_of_cutoff, case when 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) < 0 then 0 else 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) end percent_of_cutoff_not_summed, percent(summarised_ontime_count, ontime_count+la te_count, 2)percentage_of_total_by5am, percent(summarised_ontime_count+summarised_late_count, ontime_count+late_count, 2) current_percentage, ontime_count+late_count current_raw_elements, summarised_ontime_count+summarised_late_count current_summary_elements, summarised_ontime_count-summarised_late_count SUM_ELEMENTS_AT5AM, late_count late_raw_elements, ontime_count-late_count available_at_cutoff, percent(num_of_sumrows, num_of_rows, 2) accuracy from (select count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+3/24 then SGSNID end) late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+3/24 then SGSNID end) ontime_count, count (*) num_of_rows from ERICSSON_PSCORE.SGSNSTAT where datetime between trunc(sysdate)-1 and trunc(sysdate)-1/24/60)rawt, (select max(datetime)sumday, count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+7/24 then SGSNID end) summarised_late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+7/24 then SGSNID end) summarised_ontime_count, sum(entries) num_of_sumrows from ERICSSON_PSCORE.SGSNSTAT_DY where datetime = trunc(sysdate)-1)sumt union all SELECT 'ERICSSON_UTRAN.RBS_CARRIER' AS TABLENAME, sumday, CASE WHEN percent(summarised_ontime_count, (ontime_count-late_count), 2) > 100 THEN 100 ELSE percent(summarised_ontime_count, (ontime_count-late_count), 2) END percent_of_cutoff, case when 100-percent(summarised_ontime_count, (ontime_count- late_count), 2) < 0 then 0 else 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) end percent_of_cutoff_not_summed, percent(summarised_ontime_count, ontime_count+late_count, 2)percentage_of_total_by5am, percent(summarised_ontime_count+summarised_late_count, ontime_count+late_count, 2) current_percentage, ontime_count+late_count current_raw_elements, summarised_ontime_count+summarised_late_count current_summary_elements, summarised_ontime_count-summarised_late_count SUM_ELEMENTS_AT5AM, late_count late_raw_elements, ontime_count-late_count available_at_cutoff, percent(num_of_sumrows, num_of_rows, 2) accuracy from (select count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+3/24 then NODEB end) late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+3/24 then NODEB end) ontime_count, count (*) num_of_rows from ERICSSON_UTRAN.RBS_CARRIER where datetime between trunc(sysdate)-1 and trunc(sysdate)-1/24/60)rawt, (select max(datetime)sumday, count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+7/24 then NODEB end) summarised_late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+7/24 then NODEB end) summarised_ontime_count, sum(entries) num_of_sumrows from ERICSSON_UTRAN.RBS_CARRIER_DY where datetime = trunc(sysdate)-1)sumt union all SELECT 'ERICSSON_UTRAN.RBS_EDCHRES' AS TABLENAME, sumday, CASE WHEN percent(summarised_ontime_count, (ontime_count-late_count), 2) > 100 THEN 100 ELSE percent(summarised_ontime_count, (ontime_count-late_count), 2) END percent_of_cutoff, case when 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) < 0 then 0 else 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) end percent_of_cutoff_not_summed, percent(summarised_ontime_count, ontime_count+late_count, 2)percentage_of_total_by5am, percent(summarised_ontime_count+summarised_late_count, ontime_count+late_count, 2) current_percentage, ontime_count+late_count current_raw_elements, summarised_ontime_count+summarised_late_count current_summary_elements, summarised_ontime_count-summarised_late_count SUM_ELEMENTS_AT5AM, late_count late_raw_elements, ontime_count-late_count available_at_cutoff, percent(num_of_sumrows, num_of_rows, 2) accuracy from (select count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+3/24 then NODEB end) late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+3/24 then NODEB end) ontime_count, count (*) num_of_rows from ERICSSON_UTRAN.RBS_EDCHRES where datetime between trunc(sysdate)-1 and trunc(sysdate)-1/24/60)rawt, (select max(datetime)sumday, count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+7/24 then NODEB end) summarised_late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+7/24 then NODEB end) summarised_ontime_count, sum(entries) num_of_sumrows from ERICSSON_UTRAN.RBS_EDCHRES_DY where datetime = trunc(sysdate)-1)sumt union all SELECT 'ERICSSON_UTRAN.RBS_HSDSCHRES' AS TABLENAME, sumday, CASE WHEN percent(summarised_ontime_count, (ontime_count-late_count), 2) > 100 THEN 100 ELSE percent(summarised_ontime_count, (ontime_count-late_count), 2) END percent_of_cutoff, case when 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) < 0 then 0 else 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) end percent_of_cutoff_not_summed, percent(summarised_ontime_count, ontime_count+late_count, 2)percentage_of_total_by5am, percent(summarised_ontime_count+summarised_late_count, ontime_count+late_count, 2) current_percentage, ontime_count+late_count current_raw_elements, summarised_ontime_count+summarised_late_count current_summary_elements, summarised_ontime_count-summarised_late_count SUM_ELEMENTS_AT5AM, late_count late_raw_elements, ontime_count-late_count available_at_cutoff, percent(num_of_sumrows, num_of_rows, 2) accuracy from (select count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+3/24 then NODEB end) late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+3/24 then NODEB end) ontime_count, count (*) num_of_rows from ERICSSON_UTRAN.RBS_HSDSCHRES where datetime between trunc(sysdate)-1 and trunc(sysdate)-1/24/60)rawt, (select max(datetime)sumday, count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+7/24 then NODEB e nd) summarised_late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+7/24 then NODEB end) summarised_ontime_count, sum(entries) num_of_sumrows from ERICSSON_UTRAN.RBS_HSDSCHRES_DY where datetime = trunc(sysdate)-1)sumt union all SELECT 'ERICSSON_UTRAN.RNC_IURLINK' AS TABLENAME, sumday, CASE WHEN percent(summarised_ontime_count, (ontime_count-late_count), 2) > 100 THEN 100 ELSE percent(summarised_ontime_count, (ontime_count-late_count), 2) END percent_of_cutoff, case when 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) < 0 then 0 else 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) end percent_of_cutoff_not_summed, percent(summarised_ontime_count, ontime_count+late_count, 2)percentage_of_total_by5am,

Page Three


percent(summarised_ontime_count+summarised_late_count, ontime_count+late_count, 2) current_percentage, ontime_count+late_count current_raw_elements, summarised_ontime_count+summarised_late_count
current_summary_elements, summarised_ontime_count-summarised_late_count SUM_ELEMENTS_AT5AM, late_count late_raw_elements, ontime_count-late_count available_at_cutoff, percent(num_of_sumrows, num_of_rows, 2)
accuracy from (select count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+3/24 then RNC end) late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+3/24 then RNC end) ontime_count, count (*)
num_of_rows from ERICSSON_UTRAN.RNC_IURLINK where dateti me between trunc(sysdate)-1 and trunc(sysdate)-1/24/60)rawt, (select max(datetime)sunday, count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+7/24
then RNC end) summarised_late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+7/24 then RNC end) summarised_ontime_count, sum(entries) num_of_sumrows from ERICSSON_UTRAN.RNC_IURLINK_DY
where datetime = trunc(sysdate)-1)sumt union all SELECT 'ERICSSON_UTRAN.RNC_RNCFUNC' AS TABLENAME, sunday, CASE WHEN percent(summarised_ontime_count, (ontime_count-late_count), 2) > 100 THEN 100 ELSE
percent(summarised_ontime_count, (ontime_count-late_count), 2) END percent_of_cutoff, case when 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) < 0 then 0 else 100-percent(summarised_ontime_count,
(ontime_count-late_count), 2) end percent_of_cutoff_not_summed, percent(summarised_ontime_count, ontime_count+late_count, 2)percentage_of_total_by5am, percent(summarised_ontime_count+summarised_late_count,
ontime_count+late_count, 2) current_percentage, ontime_count+late_count current_raw_elements, summarised_ontime_count+summarised_late_count current_summary_elements, summarised_ontime_count-summarised_late_count
SUM_ELEMENTS_AT5AM, late_count late_raw_elements, ontime_count-late_count available_at_cutoff, percent(num_of_sumrows, num_of_rows, 2) accuracy from (select count(distinct case when trunc(datetime_ins, 'mi') >
trunc(sysdate)+3/24 then RNC end) late_c ount, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+3/24 then RNC end) ontime_count, count (*) num_of_rows from ERICSSON_UTRAN.RNC_RNCFUNC where datetime
between trunc(sysdate)-1 and trunc(sysdate)-1/24/60)rawt, (select max(datetime)sunday, count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+7/24 then RNC end) summarised_late_count, count(distinct case when
trunc(datetime_ins, 'mi') < trunc(sysdate)+7/24 then RNC end) summarised_ontime_count, sum(entries) num_of_sumrows from ERICSSON_UTRAN.RNC_RNCFUNC_DY where datetime = trunc(sysdate)-1)sumt union all SELECT
'ERICSSON_UTRAN.RNC_UCELL' AS TABLENAME, sunday, CASE WHEN percent(summarised_ontime_count, (ontime_count-late_count), 2) > 100 THEN 100 ELSE percent(summarised_ontime_count, (ontime_count-late_count), 2) END
percent_of_cutoff, case when 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) < 0 then 0 else 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) end percent_of_cutoff_not_summed,
percent(summarised_ontime_count, ontime_count+late_count, 2)percentage_of_total_by5am, percent(summarised_ontime_count+summarised_late_count, ontime_count+late_count, 2) current_percentage, ontime_count+late_count
current_raw_elements, summarised_ontime_count+summarised_late_count current_summary_elements, summarised_ontime_count-summarised_late_count SUM_ELEMENTS_AT5AM, late_count late_raw_elements, ontime_count-
late_count availabl e_at_cutoff, percent(num_of_sumrows, num_of_rows, 2) accuracy from (select count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+3/24 then RNC end) late_count, count(distinct case when
trunc(datetime_ins, 'mi') < trunc(sysdate)+3/24 then RNC end) ontime_count, count (*) num_of_rows from ERICSSON_UTRAN.RNC_UCELL where datetime between trunc(sysdate)-1 and trunc(sysdate)-1/24/60)rawt, (select
max(datetime)sunday, count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+7/24 then RNC end) summarised_late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+7/24 then RNC end)
summarised_ontime_count, sum(entries) num_of_sumrows from ERICSSON_UTRAN.RNC_UCELL_DY where datetime = trunc(sysdate)-1)sumt union all SELECT 'NORTEL_CSCORE.BASE_OM' AS TABLENAME, sunday, CASE WHEN
percent(summarised_ontime_count, (ontime_count-late_count), 2) > 100 THEN 100 ELSE percent(summarised_ontime_count, (ontime_count-late_count), 2) END percent_of_cutoff, case when 100-percent(summarised_ontime_count,
(ontime_count-late_count), 2) < 0 then 0 else 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) end percent_of_cutoff_not_summed, percent(summarised_ontime_count, ontime_count+late_count,
2)percentage_of_total_by5am, percent(summarised_ontime_count+summarised_late_count, ontime_count+late_count, 2) current_percentage, ontime_count+late_count current_raw_elements, summarised_ontime_count+summarised_
late_count current_summary_elements, summarised_ontime_count-summarised_late_count SUM_ELEMENTS_AT5AM, late_count late_raw_elements, ontime_count-late_count available_at_cutoff, percent(num_of_sumrows,
num_of_rows, 2) accuracy from (select count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+3/24 then MSC end) late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+3/24 then MSC end)
ontime_count, count (*) num_of_rows from NORTEL_CSCORE.BASE_OM where datetime between trunc(sysdate)-1 and trunc(sysdate)-1/24/60)rawt, (select max(datetime)sunday, count(distinct case when trunc(datetime_ins, 'mi') >
trunc(sysdate)+7/24 then MSC end) summarised_late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+7/24 then MSC end) summarised_ontime_count, sum(entries) num_of_sumrows from
NORTEL_CSCORE.BASE_OM_DY where datetime = trunc(sysdate)-1)sumt union all SELECT 'NORTEL_CSCORE.C7SP' AS TABLENAME, sunday, CASE WHEN percent(summarised_ontime_count, (ontime_count-late_count), 2) > 100 THEN
100 ELSE percent(summarised_ontime_count, (ontime_count-late_count), 2) END percent_of_cutoff, case when 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) < 0 then 0 else
100-percent(summarised_ontime_count, (ontime_count-late_count), 2) end percent_of_cutoff_not_summed, percent(summarised_ontime_count, ontime_count+late_count, 2)percentage_of_total_by5am, percent(summarised_ontime_c
ount+summarised_late_count, ontime_count+late_count, 2) current_percentage, ontime_count+late_count current_raw_elements, summarised_ontime_count+summarised_late_count current_summary_elements,
summarised_ontime_count-summarised_late_count SUM_ELEMENTS_AT5AM, late_count late_raw_elements, ontime_count-late_count available_at_cutoff, percent(num_of_sumrows, num_of_rows, 2) accuracy from (select
count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+3/24 then MSC end) late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+3/24 then MSC end) ontime_count, count (*) num_of_rows from
NORTEL_CSCORE.C7SP where datetime between trunc(sysdate)-1 and trunc(sysdate)-1/24/60)rawt, (select max(datetime)sunday, count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+7/24 then MSC end)
summarised_late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+7/24 then MSC end) summarised_ontime_count, sum(entries) num_of_sumrows from NORTEL_CSCORE.C7SP_DY where datetime =
trunc(sysdate)-1)sumt union all SELECT 'NORTEL_CSCORE.CCS7' AS TABLENAME, sunday, CASE WHEN percent(summarised_ontime_count, (ontime_count-late_count), 2) > 100 THEN 100 ELSE percent(summarised_ontime_count,
(ontime_count-late_count), 2) END percent_of_cutoff, case when 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) < 0 then 0 else 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) end
percent_of_cutoff_not_summed, percent(summarised_ontime_count, ontime_count+late_count, 2)percentage_of_total_by5am, percent(summarised_ontime_count+summarised_late_count, ontime_count+late_count, 2)
current_percentage, ontime_count+late_count current_raw_elements, summarised_ontime_count+summarised_late_count current_summary_elements, summarised_ontime_count-summarised_late_count SUM_ELEMENTS_AT5AM,
late_count late_raw_elements, ontime_count-late_count available_at_cutoff, percent(num_of_sumrows, num_of_rows, 2) accuracy from (select count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+3/24 then MSC end)

Page Four

late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+3/24 then MSC end) ontime_count, count (*) num_of_rows from NORTEL_CSCORE.CCS7 where datetime between trunc(sysdate)-1 and trunc(sysdate)-1/24/60)rawt, (select max(datetime)sunday, count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+7/24 then MSC end) summarised_late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+7/24 then MSC end) summarised_ontime_count, sum(entries) num_of_sumrows from NORTEL_CSCORE.CCS7_DY where datetime = trunc(sysdate)-1)sumt union all SELECT 'NORTEL_CSCORE.LOCATIONAREACODE' AS TABLENAME, sunday, CASE WHEN percent(summarised_ontime_count, (ontime_count-late_count), 2) > 100 THEN 100 ELSE percent(summarised_ontime_count, (ontime_count-late_count), 2) END percent_of_cut off, case when 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) < 0 then 0 else 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) end percent_of_cutoff_not_summed, percent(summarised_ontime_count, ontime_count+late_count, 2)percentage_of_total_by5am, percent(summarised_ontime_count+summarised_late_count, ontime_count+late_count, 2) current_percentage, ontime_count+late_count current_raw_elements, summarised_ontime_count+summarised_late_count current_summary_elements, summarised_ontime_count-summarised_late_count SUM_ELEMENTS_AT5AM, late_count late_raw_elements, ontime_count-late_count available_at_cutoff, percent(num_of_sumrows, num_of_rows, 2) accuracy from (select count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+3/24 then MSC end) late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+3/24 then MSC end) ontime_count, count (*) num_of_rows from NORTEL_CSCORE.LOCATIONAREACODE where datetime between trunc(sysdate)-1 and trunc(sysdate)-1/24/60)rawt, (select max(datetime)sunday, count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+7/24 then MSC end) summarised_late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+7/24 then MSC end) summarised_ontime_count, sum(entries) num_of_sumrows from NORTEL_CSCORE.LOCATIONAREACODE_DY where datetime = trunc(sysdate)-1)sumt union all SELECT 'NORTEL_CSCORE.MSCCP_CP' AS TABLENAME, sumd ay, CASE WHEN percent(summarised_ontime_count, (ontime_count-late_count), 2) > 100 THEN 100 ELSE percent(summarised_ontime_count, (ontime_count-late_count), 2) END percent_of_cutoff, case when 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) < 0 then 0 else 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) end percent_of_cutoff_not_summed, percent(summarised_ontime_count, ontime_count+late_count, 2)percentage_of_total_by5am, percent(summarised_ontime_count+summarised_late_count, ontime_count+late_count, 2) current_percentage, ontime_count+late_count current_raw_elements, summarised_ontime_count+summarised_late_count current_summary_elements, summarised_ontime_count-summarised_late_count SUM_ELEMENTS_AT5AM, late_count late_raw_elements, ontime_count-late_count available_at_cutoff, percent(num_of_sumrows, num_of_rows, 2) accuracy from (select count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+3/24 then MSC end) late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+3/24 then MSC end) ontime_count, count (*) num_of_rows from NORTEL_CSCORE.MSCCP_CP where datetime between trunc(sysdate)-1 and trunc(sysdate)-1/24/60)rawt, (select max(datetime)sunday, count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+7/24 then MSC end) summarised_late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+7/24 then MSC end) summarised_ontime_c ount, sum(entries) num_of_sumrows from NORTEL_CSCORE.MSCCP_CP_DY where datetime = trunc(sysdate)-1)sumt union all SELECT 'NORTEL_CSCORE.MSCCP' AS TABLENAME, sunday, CASE WHEN percent(summarised_ontime_count, (ontime_count-late_count), 2) > 100 THEN 100 ELSE percent(summarised_ontime_count, (ontime_count-late_count), 2) END percent_of_cutoff, case when 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) < 0 then 0 else 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) end percent_of_cutoff_not_summed, percent(summarised_ontime_count, ontime_count+late_count, 2)percentage_of_total_by5am, percent(summarised_ontime_count+summarised_late_count, ontime_count+late_count, 2) current_percentage, ontime_count+late_count current_raw_elements, summarised_ontime_count+summarised_late_count current_summary_elements, summarised_ontime_count-summarised_late_count SUM_ELEMENTS_AT5AM, late_count late_raw_elements, ontime_count-late_count available_at_cutoff, percent(num_of_sumrows, num_of_rows, 2) accuracy from (select count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+3/24 then MSC end) late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+3/24 then MSC end) ontime_count, count (*) num_of_rows from NORTEL_CSCORE.MSCCP where datetime between trunc(sysdate)-1 and trunc(sysdate)-1/24/60)rawt, (select max(datetime)sunday, count(distinct case when trunc(datetime_ins, 'mi') > trunc(s ysdate)+7/24 then MSC end) summarised_late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+7/24 then MSC end) summarised_ontime_count, sum(entries) num_of_sumrows from NORTEL_CSCORE.MSCCP_DY where datetime = trunc(sysdate)-1)sumt union all SELECT 'NORTEL_CSCORE.TRKGRP' AS TABLENAME, sunday, CASE WHEN percent(summarised_ontime_count, (ontime_count-late_count), 2) > 100 THEN 100 ELSE percent(summarised_ontime_count, (ontime_count-late_count), 2) END percent_of_cutoff, case when 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) < 0 then 0 else 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) end percent_of_cutoff_not_summed, percent(summarised_ontime_count, ontime_count+late_count, 2)percentage_of_total_by5am, percent(summarised_ontime_count+summarised_late_count, ontime_count+late_count, 2) current_percentage, ontime_count+late_count current_raw_elements, summarised_ontime_count+summarised_late_count current_summary_elements, summarised_ontime_count-summarised_late_count SUM_ELEMENTS_AT5AM, late_count late_raw_elements, ontime_count-late_count available_at_cutoff, percent(num_of_sumrows, num_of_rows, 2) accuracy from (select count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+3/24 then MSC end) late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+3/24 then MSC end) ontime_count, count (*) num_of_rows from NORTEL_CSCORE.TRKGRP where dateti me between trunc(sysdate)-1 and trunc(sysdate)-1/24/60)rawt, (select max(datetime)sunday, count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+7/24 then MSC end) summarised_late_count, count(distinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+7/24 then MSC end) summarised_ontime_count, sum(entries) num_of_sumrows from NORTEL_CSCORE.TRKGRP_DY where datetime = trunc(sysdate)-1)sumt union all SELECT 'NORTEL_CSCORE.VLR6' AS TABLENAME, sunday, CASE WHEN percent(summarised_ontime_count, (ontime_count-late_count), 2) > 100 THEN 100 ELSE percent(summarised_ontime_count, (ontime_count-late_count), 2) END percent_of_cutoff, case when 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) < 0 then 0 else 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) end percent_of_cutoff_not_summed, percent(summarised_ontime_count, ontime_count+late_count, 2)percentage_of_total_by5am, percent(summarised_ontime_count+summarised_late_count, ontime_count+late_count, 2) current_percentage, ontime_count+late_count current_raw_elements, summarised_ontime_count+summarised_late_count current_summary_elements, summarised_ontime_count-summarised_late_count SUM_ELEMENTS_AT5AM, late_count late_raw_elements, ontime_count-late_count available_at_cutoff, percent(num_of_sumrows, num_of_rows, 2) accuracy from (select count(distinct case when trunc(datetime_ins, 'mi') > trunc(sysdate)+3/24 then MSC end) late_count, count(d istinct case when trunc(datetime_ins, 'mi') < trunc(sysdate)+3/24 then MSC end) ontime_count, count (*) num_of_rows from NORTEL_CSCORE.VLR6 where datetime between trunc(sysdate)-1 and trunc(sysdate)-1/24/60)rawt, (select max(datetime)sunday, count(distinct case

Page Five

when trunc(datetime_ins,'mi') > trunc(sysdate)+7/24 then MSC end) summarised_late_count, count(distinct case when trunc(datetime_ins,'mi') < trunc(sysdate)+7/24 then MSC end) summarised_ontime_count, sum(entries) num_of_sumrows from NORTEL_CSCORE.VLR6_DY where datetime = trunc(sysdate)-1)sumt union all SELECT 'NORTEL_PSCORE.GSCGMM' AS TABLENAME, sunday, CASE WHEN percent(summarised_ontime_count, (ontime_count-late_count), 2) > 100 THEN 100 ELSE percent(summarised_ontime_count, (ontime_count-late_count), 2) END percent_of_cutoff, case when 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) < 0 then 0 else 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) end percent_of_cutoff_not_summed, percent(summarised_ontime_count, ontime_count+late_count, 2)percentage_of_total_by5am, percent(summarised_ontime_count+summarised_late_count, ontime_count+late_count, 2) current_percentage, ontime_count+late_count current_raw_elements, summarised_ontime_count+summarised_late_count current_summary_elements, summarised_ontime_count-summarised_late_count SUM_ELEMENTS_AT5AM, late_count late_raw_elements, ontime_count-late_count available_at_cutoff, percent(num_of_sumro ws, num_of_rows, 2) accuracy from (select count(distinct case when trunc(datetime_ins,'mi') > trunc(sysdate)+3/24 then SGSN end) late_count, count(distinct case when trunc(datetime_ins,'mi') < trunc(sysdate)+3/24 then SGSN end) ontime_count, count (*) num_of_rows from NORTEL_PSCORE.GSCGMM where datetime between trunc(sysdate)-1 and trunc(sysdate)-1/24/60)rawt, (select max(datetime)sunday, count(distinct case when trunc(datetime_ins,'mi') > trunc(sysdate)+7/24 then SGSN end) summarised_late_count, count(distinct case when trunc(datetime_ins,'mi') < trunc(sysdate)+7/24 then SGSN end) summarised_ontime_count, sum(entries) num_of_sumrows from NORTEL_PSCORE.GSCGMM_DY where datetime = trunc(sysdate)-1)sumt union all SELECT 'NORTEL_PSCORE.GSCSM_ACT' AS TABLENAME, sunday, CASE WHEN percent(summarised_ontime_count, (ontime_count-late_count), 2) > 100 THEN 100 ELSE percent(summarised_ontime_count, (ontime_count-late_count), 2) END percent_of_cutoff, case when 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) < 0 then 0 else 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) end percent_of_cutoff_not_summed, percent(summarised_ontime_count, ontime_count+late_count, 2)percentage_of_total_by5am, percent(summarised_ontime_count+summarised_late_count, ontime_count+late_count, 2) current_percentage, ontime_count+late_count current_raw_elements, summarised_ontime_count+summarised_late_count current_summary_elements, summarised_ontime_count-summarised_late_count SUM_ELEMENTS_AT5AM, late_count late_raw_elements, ontime_count-late_count available_at_cutoff, percent(num_of_sumrows, num_of_rows, 2) accuracy from (select count(distinct case when trunc(datetime_ins,'mi') > trunc(sysdate)+3/24 then SGSN end) late_count, count(distinct case when trunc(datetime_ins,'mi') < trunc(sysdate)+3/24 then SGSN end) ontime_count, count (*) num_of_rows from NORTEL_PSCORE.GSCSM_ACT where datetime between trunc(sysdate)-1 and trunc(sysdate)-1/24/60)rawt, (select max(datetime)sunday, count(distinct case when trunc(datetime_ins,'mi') > trunc(sysdate)+7/24 then SGSN end) summarised_late_count, count(distinct case when trunc(datetime_ins,'mi') < trunc(sysdate)+7/24 then SGSN end) summarised_ontime_count, sum(entries) num_of_sumrows from NORTEL_PSCORE.GSCSM_ACT_DY where datetime = trunc(sysdate)-1)sumt union all SELECT 'NORTEL_PSCORE.GSCSM' AS TABLENAME, sunday, CASE WHEN percent(summarised_ontime_count, (ontime_count-late_count), 2) > 100 THEN 100 ELSE percent(summarised_ontime_count, (ontime_count-late_count), 2) END percent_of_cutoff, case when 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) < 0 then 0 else 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) end percent_of_cutoff_not_summed, percent(summarised_ontime_count, ontime_count+late_count, 2)percentage_of_total_by5am, percent(summarised_ontime_count+summarised_late_count, ontime_count+late_count, 2) current_percentage, ontime_count+late_count current_raw_elements, summarised_ontime_count+summarised_late_count current_summary_elements, summarised_ontime_count-summarised_late_count SUM_ELEMENTS_AT5AM, late_count late_raw_elements, ontime_count-late_count available_at_cutoff, percent(num_of_sumrows, num_of_rows, 2) accuracy from (select count(distinct case when trunc(datetime_ins,'mi') > trunc(sysdate)+3/24 then SGSN end) late_count, count(distinct case when trunc(datetime_ins,'mi') < trunc(sysdate)+3/24 then SGSN end) ontime_count, count (*) num_of_rows from NORTEL_PSCORE.GSCSM where datetime between trunc(sysdate)-1 and trunc(sysdate)-1/24/60)rawt, (select max(datetime)sunday, count(distinct case when trunc(datetime_ins,'mi') > trunc(sysdate)+7/24 then SGSN end) summarised_late_count, count(distinct case when trunc(datetime_ins,'mi') < trunc(sysdate)+7/24 then SGSN end) summarised_ontime_count, sum(entries) num_of_sumrows from NORTEL_PSCORE.GSCSM_DY where datetime = trunc(sysdate)-1)sumt union all SELECT 'NORTEL_PSCORE.GSDSTATS' AS TABLENAME, sunday, CASE WHEN percent(summarised_ontime_count, (ontime_count-late_count), 2) > 100 THEN 100 ELSE percent(summarised_ontime_count, (ontime_count-late_count), 2) END percent_of_cutoff, case when 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) < 0 then 0 else 100-percent(summarised_ontime_count, (ontime_count-late_count), 2) end percent_o f_cutoff_not_summed, percent(summarised_ontime_count, ontime_count+late_count, 2)percentage_of_total_by5am, percent(summarised_ontime_count+summarised_late_count, ontime_count+late_count, 2) current_percentage, ontime_count+late_count current_raw_elements, summarised_ontime_count+summarised_late_count current_summary_elements, summarised_ontime_count-summarised_late_count SUM_ELEMENTS_AT5AM, late_count late_raw_elements, ontime_count-late_count available_at_cutoff, percent(num_of_sumrows, num_of_rows, 2) accuracy from (select count(distinct case when trunc(datetime_ins,'mi') > trunc(sysdate)+3/24 then SGSN end) late_count, count(distinct case when trunc(datetime_ins,'mi') < trunc(sysdate)+3/24 then SGSN end) ontime_count, count (*) num_of_rows from NORTEL_PSCORE.GSDSTATS where datetime between trunc(sysdate)-1 and trunc(sysdate)-1/24/60)rawt, (select max(datetime)sunday, count(distinct case when trunc(datetime_ins,'mi') > trunc(sysdate)+7/24 then SGSN end) summarised_late_count, count(distinct case when trunc(datetime_ins,'mi') < trunc(sysdate)+7/24 then SGSN end) summarised_ontime_count, sum(entries) num_of_sumrows from NORTEL_PSCORE.GSDSTATS_DY where datetime = trunc(sysdate)-1)sumt

Optimizer Plans (1:4)

```
SELECT DISTINCT E1_2.OBJECT_ID
   FROM PMCM.ELEMENT_DETAIL E1_1, PMCM.ELEMENT_DETAIL E1_2, PMCM.MARK_NETW_HIERARCHY H1,
        PMCM.ELEMENT_DETAIL E2_1, PMCM.ELEMENT_DETAIL E2_2, PMCM.MARK_NETW_HIERARCHY H2
   WHERE E1_1.OBJECT_ID = H1.PARENT_ID
     AND E1_2.OBJECT_ID = H1.OBJECT_ID
     AND E2_1.OBJECT_ID = H2.PARENT_ID
     AND E2_2.OBJECT_ID = H2.OBJECT_ID
     AND E1_1.CURRENT_IND = 'Y' AND E2_1.CURRENT_IND = 'Y'
     AND E2_1.CURRENT_IND = 'Y' AND E2_2.CURRENT_IND = 'Y'
     AND H1.CURRENT_IND = 'Y' AND H2.CURRENT_IND = 'Y'
     AND H1.HIERARCHY_TYPE = 'NETWORK' AND H2.HIERARCHY_TYPE = 'NETWORK'
     AND H1.PARENT_TYPE IN ('BSC','RNC')  AND H2.PARENT_TYPE IN ('BSC','RNC')
     AND E2_2.ELEMENT_TYPE = 'CELL' AND E1_2.ELEMENT_TYPE = 'CELL'
     AND H1.PARENT_TYPE IN ('BSC','RNC')
     AND E1_1.ELEMENT_NAME = E2_1.ELEMENT_NAME
     AND E1_1.ELEMENT_ID = E2_1.ELEMENT_ID
     AND E1_2.ELEMENT_NAME = E2_2.ELEMENT_NAME
     AND E1_2.ELEMENT_ID = E2_2.ELEMENT_ID
     AND E1_2.USEID LIKE '*%' AND E2_2.USEID NOT LIKE '*%';

-----------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name                          | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     | Pstart| Pstop |
-----------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |                               |     1 |    78 |       |    74M (40)| 50:54:42 |       |       |
|   1 |  TEMP TABLE TRANSFORMATION |                               |       |       |       |            |          |       |       |
|   2 |   LOAD AS SELECT           |                               |       |       |       |            |          |       |       |
|   3 |    PARTITION RANGE ALL     |                               |    22M|  1111M|       | 38153   (11)| 00:01:34 |     1 |    29 |
|*  4 |     TABLE ACCESS FULL      | ELEMENT_DETAIL                |    22M|  1111M|       | 38153   (11)| 00:01:34 |       |       |
|   5 |   LOAD AS SELECT           |                               |       |       |       |            |          |       |       |
|   6 |    PARTITION HASH ALL      |                               |   337K|  9231K|       |  3514   (15)| 00:00:09 |     1 |    16 |
|*  7 |     TABLE ACCESS FULL      | MARK_NETW_HIERARCHY           |   337K|  9231K|       |  3514   (15)| 00:00:09 |       |       |
|   8 |   SORT AGGREGATE           |                               |     1 |    78 |       |            |          |       |       |
|*  9 |    HASH JOIN               |                               |   927G|    65T|   534M|    74M (40)| 50:53:00 |       |       |
|  10 |     VIEW                   |                               |    22M|   277M|       | 16808   (12)| 00:00:42 |       |       |
|  11 |      TABLE ACCESS FULL     | SYS_TEMP_0FDA7485F_6A66C42E   |    22M|  1111M|       | 16808   (12)| 00       |       |       |
|* 12 |     HASH JOIN              |                               |    21G|  1272G|   534M|  1616K (43)| 01:06:04 |       |       |
|  13 |      VIEW                  |                               |    22M|   277M|       | 16808   (12)| 00:00:42 |       |       |
|  14 |       TABLE ACCESS FULL    | SYS_TEMP_0FDA7485F_6A66C42E   |    22M|  1111M|       | 16808   (12)| 0        |       |       |
|* 15 |      HASH JOIN             |                               |   476M|    23G|   524M| 97327   (22)| 00:03:59 |       |       |
|* 16 |       HASH JOIN            |                               |    10M|   401M|  8704K| 34520   (10)| 00:01:25 |       |       |
|* 17 |        HASH JOIN           |                               |   234K|  5948K|  8256K|   783   (10)| 00:00:02 |       |       |
|  18 |         VIEW               |                               |   337K|  4286K|       |   142   (14)| 00:00:01 |       |       |
|  19 |          TABLE ACCESS FULL | SYS_TEMP_0FDA74860_6A66C42E   |   337K|  3956K|       |   142   (14)| 00:00:01 |       |       |
|  20 |         VIEW               |                               |   337K|  4286K|       |   142   (14)| 00:00:01 |       |       |
|  21 |          TABLE ACCESS FULL | SYS_TEMP_0FDA74860_6A66C42E   |   337K|  3956K|       |   142   (14)| 00:00:01 |       |       |
|  22 |        VIEW                |                               |    22M|   277M|       | 16808   (12)| 00:00:42 |       |       |
|  23 |         TABLE ACCESS FULL  | SYS_TEMP_0FDA7485F_6A66C42E   |    22M|  1111M|       | 16808   (12)| 00:00:42 |       |       |
|  24 |        VIEW                |                               |    22M|   277M|       | 16808   (12)| 00:00:42 |       |       |
|  25 |         TABLE ACCESS FULL  | SYS_TEMP_0FDA7485F_6A66C42E   |    22M|  1111M|       | 16808   (12)| 0        |       |       |
-----------------------------------------------------------------------------------------------------------------------------------
```

Optimizer Plans (2:4)

```
-----------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name                          | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     | Pstart| Pstop |
-----------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                               |     1 |    78 |       |    14T(100)|999:59:59 |       |       |
|   1 |  TEMP TABLE TRANSFORMATION   |                               |       |       |       |            |          |       |       |
|   2 |   LOAD AS SELECT             |                               |       |       |       |            |          |       |       |
|   3 |    PARTITION RANGE ALL       |                               |    22M|  1111M|       | 38153   (11)| 00:01:34 |     1 |    29 |
|*  4 |     TABLE ACCESS FULL        | ELEMENT_DETAIL                |    22M|  1111M|       | 38153   (11)| 00:01:34 |       |       |
|   5 |   LOAD AS SELECT             |                               |       |       |       |            |          |       |       |
|   6 |    PARTITION HASH ALL        |                               |   337K|  9231K|       |  3514   (15)| 00:00:09 |     1 |    16 |
|*  7 |     TABLE ACCESS FULL        | MARK_NETW_HIERARCHY           |   337K|  9231K|       |  3514   (15)| 00:00:09 |       |       |
|   8 |   SORT AGGREGATE             |                               |     1 |    78 |       |            |          |       |       |
|   9 |    MERGE JOIN                |                               |   471P|    15E|       |    14T(100)|999:59:59 |       |       |
|  10 |     MERGE JOIN               |                               |    10P|   616P|       |   694G (81)|999:59:59 |       |       |
|  11 |      MERGE JOIN              |                               |   231T|    10P|       |   377G (64)|999:59:59 |       |       |
|  12 |       SORT JOIN              |                               |   334T|    11P|    28P|   377G (64)|999:59:59 |       |       |
|  13 |        MERGE JOIN CARTESIAN  |                               |   334T|    11P|       |   140G (14)|999:59:59 |       |       |
|* 14 |         HASH JOIN            |                               |   989M|    23G|   534M| 96010   (38)| 00:03:56 |       |       |
|  15 |          VIEW                |                               |    22M|   277M|       | 16808   (12)| 00:00:42 |       |       |
|  16 |           TABLE ACCESS FULL  | SYS_TEMP_0FDA7485B_6A66C42E   |    22M|  1111M|       | 16808   (12)| 00:00:42 |       |       |
|  17 |          VIEW                |                               |    22M|   277M|       | 16808   (12)| 00:00:42 |       |       |
|  18 |           TABLE ACCESS FULL  | SYS_TEMP_0FDA7485B_6A66C42E   |    22M|  1111M|       | 16808   (12)| 00:00:42 |       |       |
|  19 |         BUFFER SORT          |                               |   337K|  4286K|       |   140G (14)|999:59:59 |       |       |
|  20 |          VIEW                |                               |   337K|  4286K|       |   142   (14)| 00:00:01 |       |       |
|  21 |           TABLE ACCESS FULL  | SYS_TEMP_0FDA7485C_6A66C42E   |   337K|  3956K|       |   142   (14)| 00:00:01 |       |       |
|* 22 |       SORT JOIN              |                               |   337K|  4286K|    12M|   844   (14)| 00:00:03 |       |       |
|  23 |        VIEW                  |                               |   337K|  4286K|       |   142   (14)| 00:00:01 |       |       |
|  24 |         TABLE ACCESS FULL    | SYS_TEMP_0FDA7485C_6A66C42E   |   337K|  3956K|       |   142   (14)| 00:00:01 |       |       |
|* 25 |      SORT JOIN               |                               |    22M|   277M|   855M| 65084   (16)| 00:02:40 |       |       |
|  26 |       VIEW                   |                               |    22M|   277M|       | 16808   (12)| 00:00:42 |       |       |
|  27 |        TABLE ACCESS FULL     | SYS_TEMP_0FDA7485B_6A66C42E   |    22M|  1111M|       | 16808   (12)| 0        |       |       |
|* 28 |     SORT JOIN                |                               |    22M|   277M|   855M| 65084   (16)| 00:02:40 |       |       |
|  29 |      VIEW                    |                               |    22M|   277M|       | 16808   (12)| 00:00:42 |       |       |
|  30 |       TABLE ACCESS FULL      | SYS_TEMP_0FDA7485B_6A66C42E   |    22M|  1111M|       | 16808   (12)| 0        |       |       |
-----------------------------------------------------------------------------------------------------------------------------------
```

Optimizer Plans (3:4)

```
WITH ed AS (SELECT object_id, element_id, element_name, element_type, useid
            FROM pmcm.element_detail
            WHERE element_type = 'CELL'
            AND current_ind = 'Y'),
     mnh AS (SELECT parent_id, object_id
             FROM pmcm.mark_netw_hierarchy
             WHERE current_ind = 'Y'
             AND hierarchy_type = 'NETWORK'
             AND parent_type IN ('BSC', 'RNC'))
SELECT COUNT(*)
FROM ed e1_1, ed e1_2, ed e2_1, ed e2_2, mnh h1, mnh h2
WHERE e1_1.object_id = h1.parent_id AND   e1_2.object_id = h1.object_id
AND   e2_1.object_id = h2.parent_id AND   e2_2.object_id = h2.object_id
AND   e1_1.element_name = e2_1.element_name
AND   e1_1.element_id = e2_1.element_id
AND   e1_2.element_name = e2_2.element_name
AND   e1_2.element_id = e2_2.element_id
AND   e1_2.useid LIKE '*%'
AND   e2_2.useid NOT LIKE '*%';
```

```
-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name                | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |                     |     1 |   214 |       |   100K  (6)| 00:04:08 |
|   1 |  HASH UNIQUE                             |                     |     1 |   214 |       |   100K  (6)| 00:04:08 |
|*  2 |   HASH JOIN                              |                     |     1 |   214 |    12M|   100K  (6)| 00:04:08 |
|   3 |    PARTITION HASH ALL                    |                     |   337K|  9231K|       |  3514  (15)| 00:00:09 |
|*  4 |     TABLE ACCESS FULL                    | MARK_NETW_HIERARCHY |   337K|  9231K|       |  3514  (15)| 00:00:00 |
|*  5 |    HASH JOIN                             |                     |   207K|    36M|    22M| 95860   (6)| 00:03:56 |
|   6 |     PARTITION RANGE ALL                  |                     |   586K|    15M|       | 16233   (2)| 00:00:40 |
|   7 |      TABLE ACCESS BY LOCAL INDEX ROWID   | ELEMENT_DETAIL      |   586K|    15M|       | 16233      | ??:??:?? |
|*  8 |       INDEX SKIP SCAN                    | ED_ET_TECH_CI       |   586K|       |       | 12791   (1)| 00:00:3? |
|*  9 |     HASH JOIN                            |                     |   207K|    31M|    22M| 77982   (7)| 00:03:12 |
|  10 |      PARTITION RANGE ALL                 |                     |   586K|    15M|       | 16233   (2)| 00:00:40 |
|  11 |       TABLE ACCESS BY LOCAL INDEX ROWID  | ELEMENT_DETAIL      |   586K|    15M|       | 16233      | ??:??:?? |
|* 12 |        INDEX SKIP SCAN                   | ED_ET_TECH_CI       |   586K|       |       | 12791   (1)| 00:00:?? |
|* 13 |      HASH JOIN                           |                     |   179K|    22M|    12M| 60372   (8)| 00:02:29 |
|  14 |       PARTITION HASH ALL                 |                     |   337K|  9231K|       |  3514  (15)| 00:00:09 |
|* 15 |        TABLE ACCESS FULL                 | MARK_NETW_HIERARCHY |   337K|  9231K|       |  3514  (15)| 00:00:?? |
|* 16 |       HASH JOIN                          |                     |   184K|    17M|    10M| 55886   (8)| 00:02:18 |
|  17 |        PARTITION RANGE ALL               |                     |   184K|  9008K|       | 37137   (8)| 00:01:32 |
|* 18 |         TABLE ACCESS FULL                | ELEMENT_DETAIL      |   184K|  9008K|       | 37137   (8)| 00:01:32 |
|  19 |        PARTITION RANGE ALL               |                     |   576K|    28M|       | 17383   (8)| 00:00:43 |
|* 20 |         TABLE ACCESS BY LOCAL INDEX ROWID| ELEMENT_DETAIL      |   576K|    28M|       | 17383   (8)| ??:??:?? |
|* 21 |          INDEX SKIP SCAN                 | ED_ET_TECH_CI       |   583K|       |       | 13939   (9)| 00:00:35 |
-----------------------------------------------------------------------------------------------------------------------
```

Optimizer Plans (4:4)

```
-----------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                    | Name                    | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                             |                         |       |       |       |   264T(100)|          |
|*  1 |  VIEW                                        |                         |  156P|    15E|       |   264T (79)|999:59:59 |
|*  2 |   WINDOW SORT PUSHED RANK                    |                         |  156P|    15E|    15E|   264T (79)|999:59:59 |
|   3 |    MERGE JOIN CARTESIAN                      |                         |  156P|    15E|       |    68T (16)|999:59:59 |
|   4 |     MERGE JOIN CARTESIAN                     |                         |  220G|   205T|       |    96M (16)| 26:57:48 |
|   5 |      MERGE JOIN CARTESIAN                    |                         |  310K|   302M|       |   232  (11)| 00:00:01 |
|   6 |       MERGE JOIN CARTESIAN                   |                         |  779 |   777K|       |    22   (0)| 00:00:01 |
|   7 |        NESTED LOOPS                          |                         |       |       |       |            |          |
|   8 |         NESTED LOOPS                         |                         |    2 |  2044 |       |    20   (0)| 00:00:01 |
|   9 |          NESTED LOOPS OUTER                  |                         |    2 |  1990 |       |    18   (0)| 00:00:01 |
|  10 |           NESTED LOOPS                       |                         |    2 |  1868 |       |    17   (0)| 00:00:01 |
|  11 |            NESTED LOOPS                      |                         |    2 |  1712 |       |    15   (0)| 00:00:01 |
|  12 |             NESTED LOOPS                     |                         |    2 |  1564 |       |    13   (0)| 00:00:01 |
|  13 |              MERGE JOIN CARTESIAN            |                         |    2 |  1442 |       |    11   (0)| 00:00:01 |
|  14 |               NESTED LOOPS OUTER             |                         |    1 |   625 |       |     8   (0)| 00:00:01 |
|  15 |                NESTED LOOPS OUTER            |                         |    1 |   613 |       |     7   (0)| 00:00:01 |
|  16 |                 NESTED LOOPS                 |                         |    1 |   580 |       |     6   (0)| 00:00:01 |
|  17 |                  NESTED LOOPS OUTER          |                         |    1 |   539 |       |     5   (0)| 00:00:01 |
|  18 |                   NESTED LOOPS OUTER         |                         |    1 |   340 |       |     5   (0)| 00:00:01 |
|  19 |                    TABLE ACCESS BY INDEX ROWID| PA_STUDENT              |    1 |   316 |       |     3   (0)| 00:00:01 |
|* 20 |                     INDEX UNIQUE SCAN        | PK_STUDENT              |    1 |       |       |     2   (0)| 00:00:01 |
|  21 |                    TABLE ACCESS BY INDEX ROWID| PA_STUD_USER            |    1 |    24 |       |     2   (0)| 00:00:01 |
|* 22 |                     INDEX UNIQUE SCAN        | PK_STUD_USER            |    1 |       |       |     1   (0)| 00:00:01 |
|  23 |                   TABLE ACCESS BY INDEX ROWID| PA_ORG                  |    1 |   199 |       |     0   (0)|          |
|* 24 |                    INDEX UNIQUE SCAN         | PK_ORG                  |    1 |       |       |     0   (0)|          |
|  25 |                  TABLE ACCESS BY INDEX ROWID | PA_DOMAIN               |   13 |   533 |       |     1   (0)| 00:00:01 |
|* 26 |                   INDEX UNIQUE SCAN          | PK_DOMAIN               |    1 |       |       |     0   (0)|          |
|  27 |                 TABLE ACCESS BY INDEX ROWID  | PA_USRRF_STUD           |  100 |  3300 |       |     1   (0)| 00:00:01 |
|* 28 |                  INDEX UNIQUE SCAN           | PK_USRRF_STUD           |    1 |       |       |     0   (0)|          |
|  29 |                VIEW PUSHED PREDICATE         | PV_STUD_USER            |    1 |    12 |       |     1   (0)| 00:00:01 |
|* 30 |                 FILTER                       |                         |       |       |       |            |          |
|  31 |                  NESTED LOOPS OUTER          |                         |    1 |    22 |       |   264  (11)| 00:00:01 |
|* 32 |                   INDEX UNIQUE SCAN          | PK_STUDENT              |    1 |    10 |       |     2   (0)| 00:00:01 |
|* 33 |                   MAT_VIEW ACCESS FULL       | PV_AP_STUD_USER         |    1 |    12 |       |   262  (11)| 00:00:01 |
|  34 |                BUFFER SORT                   |                         |    2 |   192 |       |    10   (0)| 00:00:01 |
|  35 |                 TABLE ACCESS BY INDEX ROWID  | PA_CPNT_COMPLIANCE_DATA |    2 |   192 |       |     3   (0)| 00:00:01 |
|* 36 |                  INDEX RANGE SCAN            | IX_CPNT_CD__EVTHST      |    2 |       |       |     1   (0)| 00:00:01 |
|  37 |               TABLE ACCESS BY INDEX ROWID    | PA_CPNT_TYPE            |    1 |    61 |       |     1   (0)| 00:00:01 |
|* 38 |                INDEX UNIQUE SCAN             | PK_CPNT_TYPE            |    1 |       |       |     0   (0)|          |
|  39 |              TABLE ACCESS BY INDEX ROWID     | PA_RQMT_TYPE            |    1 |    74 |       |     1   (0)| 00:00:01 |
|* 40 |               INDEX UNIQUE SCAN              | PK_RQMT_TYPE            |    1 |       |       |     0   (0)|          |
|  41 |              TABLE ACCESS BY INDEX ROWID     | PA_CMPL_STAT            |    1 |    78 |       |     1   (0)| 00:00:01 |
|* 42 |               INDEX UNIQUE SCAN              | PK_CMPL_STAT            |    1 |       |       |     0   (0)|          |
|  43 |             TABLE ACCESS BY INDEX ROWID      | PA_QUAL                 |    1 |    61 |       |     1   (0)| 00:00:01 |
|* 44 |              INDEX UNIQUE SCAN               | PK_QUAL                 |    1 |       |       |     0   (0)|          |
|* 45 |            INDEX UNIQUE SCAN                 | PK_CPNT                 |    1 |       |       |     0   (0)|          |
|  46 |           TABLE ACCESS BY INDEX ROWID        | PA_CPNT                 |    1 |    27 |       |     1   (0)| 00:00:01 |
|  47 |         BUFFER SORT                          |                         |  399 |       |       |    21   (0)| 00:00:01 |
|  48 |          INDEX FAST FULL SCAN                | PK_USRRF_STUD           |  399 |       |       |     1   (0)| 00:00:01 |
|  49 |        BUFFER SORT                           |                         |  399 |       |       |   231  (11)| 00:00:01 |
|  50 |         INDEX FAST FULL SCAN                 | PK_USRRF_STUD           |  399 |       |       |     0   (0)|          |
|  51 |       BUFFER SORT                            |                         |  710K|       |       |    96M (16)| 26:57:48 |
|  52 |        INDEX FAST FULL SCAN                  | IX_STUD_USER__STUDENT   |  710K|       |       |   309  (16)| 00:00:01 |
|  53 |      BUFFER SORT                             |                         |  710K|       |       |   264T (79)|999:59:59 |
|  54 |       INDEX FAST FULL SCAN                   | IX_STUD_USER__STUDENT   |  710K|       |       |   309  (16)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------------------------
```

Poorly Written Applications

- There's nothing wrong with the SQL ... but there is definitely something wrong

SQL ordered by Executions

- Total Executions: 29,717,627
- Captured SQL account for 77.4% of Total

Executions	Rows Processed	Rows per Exec	CPU per Exec (s)	Elap per Exec (s)	SQL Id	SQL Module	SQL Text
10,128,178	2,506,529	0.25	0.00	0.00	932srzq1krc33	ASN_07B_DIP(004110016)	SELECT NE_TIMEZONE FROM CMPM.E...
7,576,759	7,579,197	1.00	0.00	0.00	1h698sb62un99	asci_56_RANAPProtocolStats(01611000E)	SELECT DISTINCT NE_TIMEZONE FR...
3,914,621	3,848,268	0.98	0.00	0.00	5tbzdddgguu8cc	asci_56_RANAPProtocolStats(01611000E)	SELECT SYS_VERSION FROM CMPM.T...
311,645	311,604	1.00	0.00	0.00	7qtztzv329wq0		select c.name, u.name from co...
301,428	301,325	1.00	0.00	0.00	36s446f9cnwhw		SELECT C.NAME FROM COL$ C WHER...
200,692	200,669	1.00	0.00	0.00	4vs91dcv7u1p6	OMS	insert into sys.aud$(sessioni...
65,044	65,035	1.00	0.00	0.00	fz9xwpt2cvt0k		SELECT par_type, param_clob, ...
64,949	3,945,482	60.75	0.00	0.00	f5ra7dru5fk5n	XML_P7R_RNC_RCS(003110008)	SELECT NAME, PATH, READ, WR...
64,801	64,807	1.00	0.00	0.00	fhzj09a7fnrnb	XML_V7I_IN_LP_DC(00811000V)	SELECT DBTIMEZONE, LENGTH(DBT...
64,632	64,542	1.00	0.00	0.00	15jnrrb6016nd	XML_V7I_IN_LP_DC(00811000V)	SELECT SESSIONTIMEZONE, LENGT...

```
SELECT /*+ RESULT_CACHE */ srvr_id
FROM (
  SELECT srvr_id, SUM(cnt) SUMCNT
  FROM (
    SELECT DISTINCT srvr_id, 1 AS CNT
    FROM servers
    UNION ALL
    SELECT DISTINCT srvr_id, 1
    FROM serv_inst)
  GROUP BY srvr_id)
WHERE sumcnt = 2;
```

more examples: www.morganslibrary.org/reference/pkgs/dbms_result_cache.html

Optimizer Settings

- Default Oracle install favors data warehouse not OLTP

```
NAME                                     TYPE         VALUE
---------------------------------------- ------------ ------------
optimizer_adaptive_features              boolean      TRUE
optimizer_adaptive_reporting_only        boolean      FALSE
optimizer_capture_sql_plan_baselines     boolean      FALSE
optimizer_dynamic_sampling               integer      2
optimizer_features_enable                string       12.1.0.2
optimizer_index_caching                  integer      0
optimizer_index_cost_adj                 integer      100
optimizer_mode                           string       ALL_ROWS
optimizer_secure_view_merging            boolean      TRUE
optimizer_use_invisible_indexes          boolean      FALSE
optimizer_use_pending_statistics         boolean      FALSE
optimizer_use_sql_plan_baselines         boolean      TRUE
```

```
SQL> show parameter optimizer_mode

NAME                                 TYPE         VALUE
------------------------------------ ------------ -------
--
optimizer_mode                       string
ALL_ROWS

SQL> ALTER SYSTEM SET OPTIMIZER_MODE='FIRST_ROWS_10';

System altered.

SQL> show parameter optimizer_mode

NAME                         TYPE         VALUE
---------------------------- ------------ ---------------
optimizer_mode               string       FIRST_ROWS_10
```

Best for Data Warehouse

Best for OLTP

more examples: www.morganslibrary.org/reference/startup_parms.html

Optimizer Mode

- Establishes the default behavior for choosing an optimization approach for the instance
  - Maximum Throughput (Data Warehouse)

```
SQL> show parameter optimizer_mode

NAME                                 TYPE        VALUE
------------------------------------ ----------- ---------
optimizer_mode                       string      ALL_ROWS
```

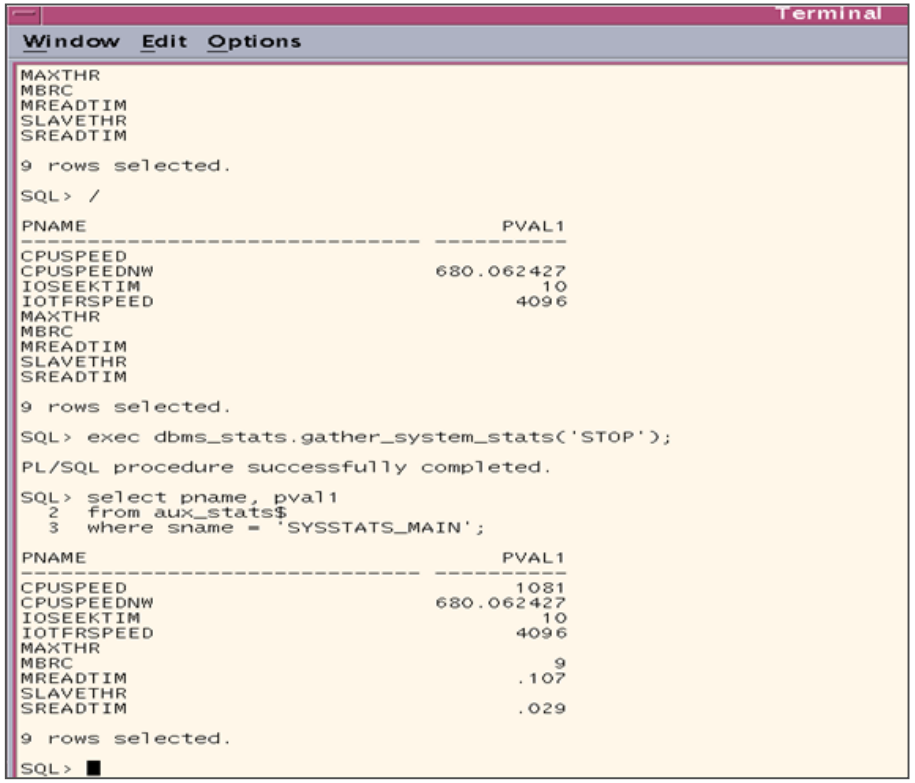
  Best Response Time (loading a web page)

```
SQL> ALTER SYSTEM SET OPTIMIZER_MODE='FIRST_ROWS_10';

System altered.

SQL> show parameter optimizer_mode

NAME                          TYPE        VALUE
----------------------------- ----------- --------------
optimizer_mode                string      FIRST_ROWS_10
```

Optimizer Statistics and Preferences

Optimizer Statistics

- Gather System Stats
- Gather Fixed Object Stats
- Gather Dictionary Stats
- Gather Table Stats
- Gather Column Stats
- Gather Index Stats
- Gather Extended Stats
- Stat Generation
  - Copy Stats
  - Set Stats
  - Fixing Stats




```
SQL> select type, count(*)
  2  from v$fixed_table
  3  group by type
  4  order by 1;

TYPE    COUNT(*)
----- ----------
TABLE       1144
VIEW        1261

SQL> select name
  2  from v$fixed_table
  3  where rownum < 11;

NAME
--------------------------
X$KQFTA
X$KQFVI
X$KQFVT
X$KQFDT
X$KQFCO
X$KQFOPT
X$KYWMPCTAB
X$KYWMWRCTAB
X$KYWMCLTAB
X$KYWMNF
```

more examples: www.morganslibrary.org/reference/system_stats.html

Gathering Fixed Object Stats

- Gathers stats on "fixed objects"
  - fixed_obj$
  - v$fixed_table

```
dbms_stats.gather_fixed_objects_stats (
stattab       IN VARCHAR2 DEFAULT NULL,
statid        IN VARCHAR2 DEFAULT NULL,
statown       IN VARCHAR2 DEFAULT NULL,
no_invalidate IN BOOLEAN  DEFAULT
              to_no_invalidate_type(get_param('NO_INVALIDATE')));
```

```
SQL> SELECT type, count(*)
  2  FROM v$fixed_table
  3  GROUP BY type;

TYPE    COUNT(*)
----- ----------
TABLE         1144
VIEW          1261

SQL> select name
  2  from v$fixed_table
  3  where rownum < 11;

NAME
--------------------------
X$KQFTA
X$KQFVI
X$KQFVT
X$KQFDT
X$KQFCO
X$KQFOPT
X$KYWMPCTAB
X$KYWMWRCTAB
X$KYWMCLTAB
X$KYWMNF
```

System Statistics

- System statistics are collected by the DBMS_STATS package only when the procedure is manually executed
- If you do not have system stats collected then the optimizer has no information about the server and storage environment

```
SQL> exec dbms_stats.gather_system_stats('INTERVAL', 15);

SQL> SELECT * FROM sys.aux_stats$;

SNAME           PNAME                PVAL1 PVAL2
--------------- --------------- ---------- ----------------
SYSSTATS_INFO   STATUS                     COMPLETED
SYSSTATS_INFO   DSTART                     05-27-2015 09:45
SYSSTATS_INFO   DSTOP                      05-27-2015 09:51
SYSSTATS_INFO   FLAGS                   0
SYSSTATS_MAIN   CPUSPEEDNW           3010
SYSSTATS_MAIN   IOSEEKTIM              10
SYSSTATS_MAIN   IOTFRSPEED           4096
SYSSTATS_MAIN   SREADTIM            3.862
SYSSTATS_MAIN   MREADTIM            1.362
SYSSTATS_MAIN   CPUSPEED             2854
SYSSTATS_MAIN   MBRC                   17
SYSSTATS_MAIN   MAXTHR
SYSSTATS_MAIN   SLAVETHR
```

Processing Rates (1:2)

- Besides the amount of work the optimizer also needs to know the HW characteristics of the system to understand how much time is needed to complete that amount of work
- Consequently, the HW characteristics describe how much work a single process can perform on that system, these are expressed as bytes per second and rows per second and are called processing rates
- As they indicate a system's capability it means you will need fewer processes (which means less DOP) for the same amount of work as these rates go higher; the more powerful a system is, the less resources you need to process the same statement in the same amount of time
- Processing rates are collected manually

```
SQL> exec dbms_stats.gather_processing_rate('START', 20);

SQL> SELECT operation_name, manual_value, calibration_value, default_value
  2  FROM v$optimizer_processing_rate
  3  ORDER BY 1;
```

Processing Rates (2:2)

- Processing Rate collection is new as of version 12cR1

```
OPERATION_NAME            MANUAL_VAL CALIBRATIO DEFAULT_VA
------------------------- ---------- ---------- ----------
AGGR                                            1000.00000
ALL                                              200.00000
CPU                                              200.00000
CPU_ACCESS                                       200.00000
CPU_AGGR                                         200.00000
CPU_BYTES_PER_SEC                               1000.00000
CPU_FILTER                                       200.00000
CPU_GBY                                          200.00000
CPU_HASH_JOIN                                    200.00000
CPU_IMC_BYTES_PER_SEC                           2000.00000
CPU_IMC_ROWS_PER_SEC                            2000000.00
CPU_JOIN                                         200.00000
CPU_NL_JOIN                                      200.00000
CPU_RANDOM_ACCESS                                200.00000
CPU_ROWS_PER_SEC                             1000000.00000
CPU_SEQUENTIAL_ACCESS                            200.00000
CPU_SM_JOIN                                      200.00000
CPU_SORT                                         200.00000
HASH                                             200.00000
IO                                               200.00000
IO_ACCESS                                        200.00000
IO_BYTES_PER_SEC                                 200.00000
IO_IMC_ACCESS                                   1000.00000
IO_RANDOM_ACCESS                                 200.00000
IO_ROWS_PER_SEC                              1000000.00000
IO_SEQUENTIAL_ACCESS                             200.00000
MEMCMP                                           500.00000
MEMCPY                                          1000.00000

SQL> exec dbms_stats.set_processing_rate('IO', 100);
```

Getting Optimizer Preferences

- DBMS_STATS.GET_PREFS returns the current preference for the identified parameter
- Parameters for which preferences can be set are CASCADE, DEGREE, ESTIMATE_PERCENT, METHOD_OPT, and NO_INVALIDATE

```
dbms_stats.get_prefs(
pname    IN VARCHAR2,
ownname  IN VARCHAR2 DEFAULT NULL,
tabname  IN VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

```
SQL> SELECT dbms_stats.get_prefs('METHOD_OPT', USER)
  2  FROM dual;

DBMS_STATS.GET_PREFS('METHOD_OPT',USER)
-----------------------------------------------
FOR ALL COLUMNS SIZE AUTO

SQL> SELECT dbms_stats.get_prefs('CASCADE', USER, 'SERVERS')
  2  FROM dual;

DBMS_STATS.GET_PREFS('CASCADE',USER,'SERVERS')
-----------------------------------------------------------
DBMS_STATS.AUTO_CASCADE
```

Setting Optimizer Preferences

- Optimizer preferences can bet set at the GLOBAL, DATABASE, SCHEMA, and TABLE levels
- Syntax

```
dbms_stats.set_table_prefs(
ownname IN VARCHAR2,
tabname IN VARCHAR2,
pname   IN VARCHAR2,
pvalue  IN VARCHAR2);
```

- Examples

```
exec dbms_stats.set_table_prefs(USER, 'SERVERS', 'CASCADE', 'DBMS_STATS.AUTO_CASCADE');

exec dbms_stats.set_table_prefs(USER, 'SERVERS', 'ESTIMATE_PERCENT','90');

exec dbms_stats.set_table_prefs(USER, 'SERVERS', 'DEGREE','8');
```

- The larger the table the smaller the sample size (ESTIMATE_PERCENT) and the larger the degree of parallelism (DEGREE) to consider

Create Extended Stats

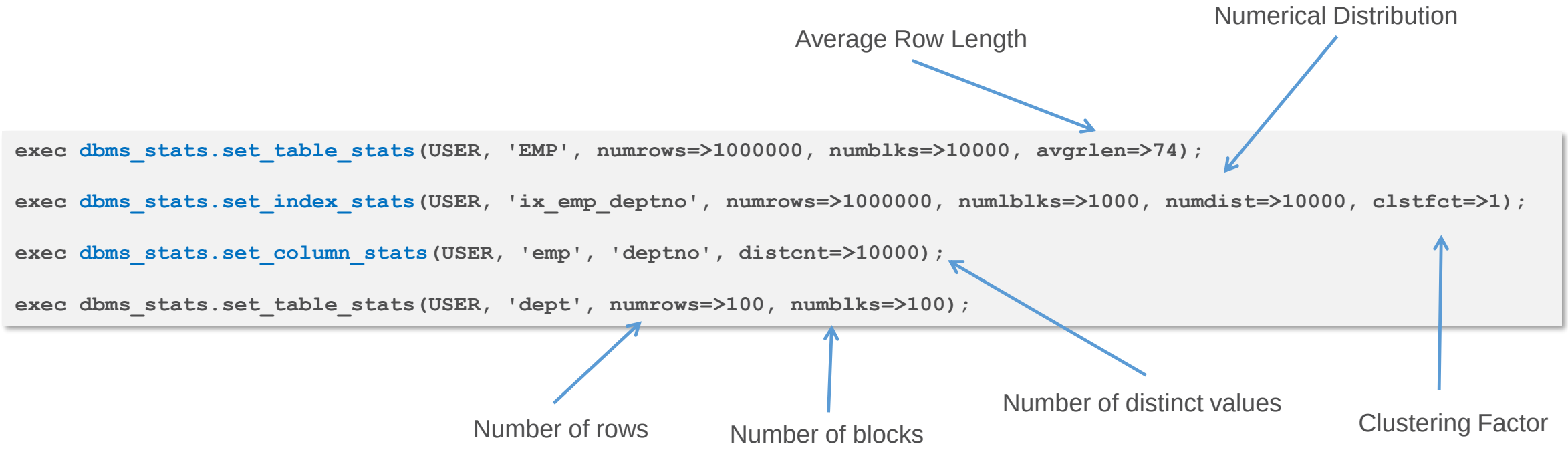
- Where WHERE clause predicates utilize more than a single table column collect extended stats
- Allows for the creation of stats that relate to a data distribution across multiple columns in a single table

```
SELECT dbms_stats.create_extended_stats(USER, 'SERV_INST', '(srvr_id, si_status)')
FROM dual;
```

more examples: www.morganslibrary.org/reference/pkgs/dbms_stats.html

Manufacture Optimizer Statistics

- Creating a new table or partition?
  - If you know approximately what will be in it when it is full set the statistics when you create it
  - If working in a DEV or TEST environment set or import stats to make these environments "look" more like production



more examples: www.morganslibrary.org/reference/pkgs/dbms_stats.html

Explain Plan and Diagnostics

Root Cause Analysis by Sophisticated Guessing

- Check data dictionary for collected stats
- Explain Plans
  - Very few people can read them ... I will prove it next
- AWR Reports
  - Data <> Information
  - If you want value from AWR ... create **AWR Difference** reports with **DBMS_WORKLOAD_REPOSITORY.AWR_DIFF_REPORT_HTML**
- ADDM Difference Reports
- ASH Reports
- SQL Tuning Advisor
- SQL Trace and TKPROF
- Baselines and Evolving Baselines
- DBMS_TRACE, DBMS_MONITOR, TRCSESS, TRACE ANALYZER, SQLTXPLAIN

Reading Explain Plans

- Very few people can read an explain plan
- Here are two plans from identical tables with identical stats
- Which one would you rely on?

The SQL Statement

```
SELECT srvr_id
FROM servers
INTERSECT
SELECT srvr_id
FROM serv_inst;
```

Database 11gR2

```
-------------------------------------------------------------------------------
| Id | Operation              | Name         | Rows | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT       |              |  141 | 4560  |     6  (84)|
|  1 |  INTERSECTION          |              |      |       |            |
|  2 |   SORT UNIQUE NOSORT   |              |  141 |  564  |     2  (50)|
|  3 |    INDEX FULL SCAN     | PK_SERVERS   |  141 |  564  |     1   (0)|
|  4 |   SORT UNIQUE          |              |  999 | 3996  |     4  (25)|
|  5 |    INDEX FAST FULL SCAN| IX_SERV_INST |  999 | 3996  |     3   (0)|
-------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------
| Id  | Operation              | Name         | Rows | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |              |  141 |  4560 |    20  (10)|
|   1 |  INTERSECTION          |              |      |       |            |
|   2 |   SORT UNIQUE          |              |  141 |   564 |    10  (10)|
|   3 |    TABLE ACCESS FULL   | SERVERS      |  141 |   564 |     9   (0)|
|   4 |   SORT UNIQUE          |              |  999 |  3996 |    10  (10)|
|   5 |    TABLE ACCESS FULL   | SERV_INST    |  999 |  3996 |     9   (0)|
-------------------------------------------------------------------------------
```

Database 12gR1

```
SQL> SELECT table_name, blocks
  2  FROM user_tables
  3* WHERE table_name IN ('SERVERS', 'SERV_INST');

TABLE_NAME                     BLOCKS
------------------------------ ----------
SERVERS                            28
SERV_INST                          28
```

more examples: www.morganslibrary.org/reference/explain_plan.html

This Adds To The Confusion

```
SQL> SELECT blocks
  2  FROM dba_tables
  3  WHERE owner = 'UWCLASS'
  4  AND table_name = 'SERVERS';

    BLOCKS
----------
        28

SQL> SELECT blocks
  2  FROM dba_segments
  3  WHERE owner = 'UWCLASS'
  4  AND segment_name = 'SERVERS';

    BLOCKS
----------
        32
```

Creating Explain Plans (3:3)

- The challenge is to find the the most efficient way to access the data
- Which of these statements is best?

```
SELECT srvr_id
FROM servers
INTERSECT
SELECT srvr_id
FROM serv_inst;
```

```
SELECT srvr_id
FROM servers
WHERE srvr_id IN (
  SELECT srvr_id
  FROM serv_inst);
```

```
SELECT srvr_id
FROM servers s
WHERE EXISTS (
  SELECT srvr_id
  FROM serv_inst i
  WHERE s.srvr_id = i.srvr_id);
```

```
SELECT DISTINCT s.srvr_id
FROM servers s, serv_inst i
WHERE s.srvr_id = i.srvr_id;
```

Creating The Plan

```
EXPLAIN PLAN FOR
SELECT srvr_id
FROM servers s
WHERE EXISTS (
  SELECT srvr_id
  FROM serv_inst i
  WHERE s.srvr_id = i.srvr_id);
```

Creating The Plan Report

```
SQL> EXPLAIN PLAN FOR
  2  SELECT srvr_id
  3  FROM servers s
  4  WHERE EXISTS (
  5    SELECT srvr_id
  6    FROM serv_inst i
  7    WHERE s.srvr_id = i.srvr_id);
Explained.

SQL> SELECT * FROM TABLE(dbms_xplan.display);

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
Plan hash value: 2840037858
------------------------------------------------------------------------------------------
| Id  | Operation              | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |              |    11 |   286 |     4  (25)| 00:00:01 |
|   1 |  NESTED LOOPS          |              |    11 |   286 |     4  (25)| 00:00:01 |
|   2 |   SORT UNIQUE          |              |   999 | 12987 |     3   (0)| 00:00:01 |
|   3 |    INDEX FAST FULL SCAN| PK_SERV_INST |   999 | 12987 |     3   (0)| 00:00:01 |
|*  4 |   INDEX UNIQUE SCAN    | PK_SERVERS   |     1 |    13 |     0   (0)| 00:00:01 |
------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("S"."SRVR_ID"="I"."SRVR_ID")

Note
-----
   - dynamic sampling used for this statement
```

Reading Explain Plans (1:6)

- Read from the most indented out and from the bottom to the top
- Sum costs with similar indents in the indent group
- Use the CPU Percentage to determine the portion of the cost that is CPU
- The difference is the disk I/O cost

Reading Explain Plans (2:6)

```
SQL> EXPLAIN PLAN FOR
  2  SELECT srvr_id
  3  FROM servers s
  4  WHERE EXISTS (
  5    SELECT srvr_id
  6    FROM serv_inst i
  7    WHERE s.srvr_id = i.srvr_id);
Explained.

SQL> SELECT * FROM TABLE(dbms_xplan.display);
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
Plan hash value: 2840037858
------------------------------------------------------------------------------------------
| Id  | Operation              | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |              |    11 |   286 |     4  (25)| 00:00:01 |
|   1 |  NESTED LOOPS          |              |    11 |   286 |     4  (25)| 00:00:01 |
|   2 |   SORT UNIQUE          |              |   999 | 12987 |     3   (0)| 00:00:01 |
|   3 |    INDEX FAST FULL SCAN| PK_SERV_INST |   999 | 12987 |     3   (0)| 00:00:01 |
|*  4 |   INDEX UNIQUE SCAN    | PK_SERVERS   |     1 |    13 |     0   (0)| 00:00:01 |
------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("S"."SRVR_ID"="I"."SRVR_ID")

Note
-----
   - dynamic sampling used for this statement
```

1. Start with the most indented: Read 999 rows, ~13KB from the SERV_INST table's primary key index
2. Since there is no CPU percentage the cost indicates it will read 3 blocks

Reading Explain Plans (3:6)

```
SQL> EXPLAIN PLAN FOR
  2  SELECT srvr_id
  3  FROM servers s
  4  WHERE EXISTS (
  5    SELECT srvr_id
  6    FROM serv_inst i
  7    WHERE s.srvr_id = i.srvr_id);
Explained.

SQL> SELECT * FROM TABLE(dbms_xplan.display);
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
Plan hash value: 2840037858
------------------------------------------------------------------------------------------
| Id  | Operation              | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |              |    11 |   286 |     4  (25)| 00:00:01 |
|   1 |  NESTED LOOPS          |              |    11 |   286 |     4  (25)| 00:00:01 |
|   2 |   SORT UNIQUE          |              |   999 | 12987 |     3   (0)| 00:00:01 |
|   3 |    INDEX FAST FULL SCAN| PK_SERV_INST |   999 | 12987 |     3   (0)| 00:00:01 |
|*  4 |   INDEX UNIQUE SCAN    | PK_SERVERS   |     1 |    13 |     0   (0)| 00:00:01 |
------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("S"."SRVR_ID"="I"."SRVR_ID")

Note
-----
   - dynamic sampling used for this statement
```

3. Sort for the query of the PK_SERV_INST index for unique values

Reading Explain Plans (4:6)

```
SQL> EXPLAIN PLAN FOR
  2  SELECT srvr_id
  3  FROM servers s
  4  WHERE EXISTS (
  5    SELECT srvr_id
  6    FROM serv_inst i
  7    WHERE s.srvr_id = i.srvr_id);
Explained.

SQL> SELECT * FROM TABLE(dbms_xplan.display);
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
Plan hash value: 2840037858
------------------------------------------------------------------------------------------
| Id  | Operation              | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |              |    11 |   286 |     4  (25)| 00:00:01 |
|   1 |  NESTED LOOPS          |              |    11 |   286 |     4  (25)| 00:00:01 |
|   2 |   SORT UNIQUE          |              |   999 | 12987 |     3   (0)| 00:00:01 |
|   3 |    INDEX FAST FULL SCAN| PK_SERV_INST |   999 | 12987 |     3   (0)| 00:00:01 |
|*  4 |   INDEX UNIQUE SCAN    | PK_SERVERS   |     1 |    13 |     0   (0)| 00:00:01 |
------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("S"."SRVR_ID"="I"."SRVR_ID")

Note
-----
   - dynamic sampling used for this statement
```

4. Read one row, 13 bytes from the SERVER table's primary key index: The cost is negligible

Reading Explain Plans (5:6)

```
SQL> EXPLAIN PLAN FOR
  2  SELECT srvr_id
  3  FROM servers s
  4  WHERE EXISTS (
  5    SELECT srvr_id
  6    FROM serv_inst i
  7    WHERE s.srvr_id = i.srvr_id);
Explained.

SQL> SELECT * FROM TABLE(dbms_xplan.display);
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
Plan hash value: 2840037858
------------------------------------------------------------------------------------------
| Id  | Operation              | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |              |    11 |   286 |     4  (25)| 00:00:01 |
|   1 |  NESTED LOOPS          |              |    11 |   286 |     4  (25)| 00:00:01 |
|   2 |   SORT UNIQUE          |              |   999 | 12987 |     3   (0)| 00:00:01 |
|   3 |    INDEX FAST FULL SCAN| PK_SERV_INST |   999 | 12987 |     3   (0)| 00:00:01 |
|*  4 |   INDEX UNIQUE SCAN    | PK_SERVERS   |     1 |    13 |     0   (0)| 00:00:01 |
------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("S"."SRVR_ID"="I"."SRVR_ID")

Note
-----
   - dynamic sampling used for this statement
```

5. Use a NESTED LOOP to join the results of the two index queries
6. The cost after this operation will bes 4 of which 25% is CPU (3+1=4)

Reading Explain Plans (6:6)

```
SQL> EXPLAIN PLAN FOR
  2  SELECT srvr_id
  3  FROM servers s
  4  WHERE EXISTS (
  5    SELECT srvr_id
  6    FROM serv_inst i
  7    WHERE s.srvr_id = i.srvr_id);
Explained.

SQL> SELECT * FROM TABLE(dbms_xplan.display);
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
Plan hash value: 2840037858
------------------------------------------------------------------------------------------
| Id  | Operation              | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |              |    11 |   286 |     4  (25)| 00:00:01 |
|   1 |  NESTED LOOPS          |              |    11 |   286 |     4  (25)| 00:00:01 |
|   2 |   SORT UNIQUE          |              |   999 | 12987 |     3   (0)| 00:00:01 |
|   3 |    INDEX FAST FULL SCAN| PK_SERV_INST |   999 | 12987 |     3   (0)| 00:00:01 |
|*  4 |   INDEX UNIQUE SCAN    | PK_SERVERS   |     1 |    13 |     0   (0)| 00:00:01 |
------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("S"."SRVR_ID"="I"."SRVR_ID")

Note
-----
   - dynamic sampling used for this statement
```

7. The result returned to the end-user will be 11 rows (286 bytes)

A More Complex Explain Plan (1:8)

```
-------------------------------------------------------------------------------
| Id | Operation                    | Name          | Rows | Bytes |Cost(%CPU)|
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT             |               |    1 |    17 |    9 (45)|
|  1 |  HASH UNIQUE                 |               |    1 |    17 |    9 (45)|
|* 2 |   HASH JOIN ANTI             |               |  140 |  2380 |    8 (38)|
|  3 |    INDEX FULL SCAN           | PK_SERVERS    |  141 |   564 |    1  (0)|
|  4 |    VIEW                      | VW_NSO_1      |  141 |  1833 |    6 (34)|
|  5 |     MINUS                    |               |      |       |          |
|  6 |      SORT UNIQUE             |               |  141 |   564 |          |
|  7 |       INDEX FULL SCAN        | PK_SERVERS    |  141 |   564 |    1  (0)|
|  8 |      SORT UNIQUE             |               |  999 |  3996 |          |
|  9 |       INDEX FAST FULL SCAN   | IX_SERV_INST  |  999 |  3996 |    3  (0)|
-------------------------------------------------------------------------------
```

1. Read 999 rows, about 4K of disk, which is 8 blocks
2. Sort the query result for unique values

A More Complex Explain Plan (2:8)

```
-------------------------------------------------------------------------------
| Id | Operation                    | Name          | Rows | Bytes |Cost(%CPU)|
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT             |               |    1 |    17 |    9 (45)|
|  1 |  HASH UNIQUE                 |               |    1 |    17 |    9 (45)|
|* 2 |   HASH JOIN ANTI             |               |  140 |  2380 |    8 (38)|
|  3 |    INDEX FULL SCAN           | PK_SERVERS    |  141 |   564 |    1  (0)|
|  4 |    VIEW                      | VW_NSO_1      |  141 |  1833 |    6 (34)|
|  5 |     MINUS                    |               |      |       |          |
|  6 |      SORT UNIQUE             |               |  141 |   564 |          |
|  7 |       INDEX FULL SCAN        | PK_SERVERS    |  141 |   564 |    1  (0)|
|  8 |      SORT UNIQUE             |               |  999 |  3996 |          |
|  9 |       INDEX FAST FULL SCAN   | IX_SERV_INST  |  999 |  3996 |    3  (0)|
-------------------------------------------------------------------------------
```

3. Read 141 rows, about 0.5K of disk, which is 1 block
4. Sort the query result for unique values

A More Complex Explain Plan (3:8)

```
-------------------------------------------------------------------------------
| Id | Operation                    | Name          | Rows | Bytes |Cost(%CPU)|
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT             |               |    1 |    17 |    9 (45)|
|  1 |  HASH UNIQUE                 |               |    1 |    17 |    9 (45)|
|* 2 |   HASH JOIN ANTI             |               |  140 |  2380 |    8 (38)|
|  3 |    INDEX FULL SCAN           | PK_SERVERS    |  141 |   564 |    1  (0)|
|  4 |    VIEW                      | VW_NSO_1      |  141 |  1833 |    6 (34)|
|  5 |     MINUS                    |               |      |       |          |
|  6 |      SORT UNIQUE             |               |  141 |   564 |          |
|  7 |       INDEX FULL SCAN        | PK_SERVERS    |  141 |   564 |    1  (0)|
|  8 |      SORT UNIQUE             |               |  999 |  3996 |          |
|  9 |       INDEX FAST FULL SCAN   | IX_SERV_INST  |  999 |  3996 |    3  (0)|
-------------------------------------------------------------------------------
```

5. Subtract the result of the IX_SERV_INST query from the result of the PK_SERVERS query

A More Complex Explain Plan (4:8)

```
-------------------------------------------------------------------------------
| Id | Operation                    | Name          | Rows | Bytes |Cost(%CPU)|
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT             |               |    1 |    17 |    9 (45)|
|  1 |  HASH UNIQUE                 |               |    1 |    17 |    9 (45)|
|* 2 |   HASH JOIN ANTI             |               |  140 |  2380 |    8 (38)|
|  3 |    INDEX FULL SCAN           | PK_SERVERS    |  141 |   564 |    1  (0)|
|  4 |    VIEW                      | VW_NSO_1      |  141 |  1833 |    6 (34)|
|  5 |     MINUS                    |               |      |       |          |
|  6 |      SORT UNIQUE             |               |  141 |   564 |          |
|  7 |       INDEX FULL SCAN        | PK_SERVERS    |  141 |   564 |    1  (0)|
|  8 |      SORT UNIQUE             |               |  999 |  3996 |          |
|  9 |       INDEX FAST FULL SCAN   | IX_SERV_INST  |  999 |  3996 |    3  (0)|
-------------------------------------------------------------------------------
```

6. Materialize the result of the subtraction as a view
7. The cost up to now has been 4 (3+1). Now the cost is 6 of which 1/3 (2) is CPU

A More Complex Explain Plan (5:8)

```
-------------------------------------------------------------------------------
| Id | Operation                    | Name          | Rows | Bytes |Cost(%CPU)|
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT             |               |    1 |    17 |    9 (45)|
|  1 |  HASH UNIQUE                 |               |    1 |    17 |    9 (45)|
|* 2 |   HASH JOIN ANTI             |               |  140 |  2380 |    8 (38)|
|  3 |    INDEX FULL SCAN           | PK_SERVERS    |  141 |   564 |    1  (0)|
|  4 |    VIEW                      | VW_NSO_1      |  141 |  1833 |    6 (34)|
|  5 |     MINUS                    |               |      |       |          |
|  6 |      SORT UNIQUE             |               |  141 |   564 |          |
|  7 |       INDEX FULL SCAN        | PK_SERVERS    |  141 |   564 |    1  (0)|
|  8 |      SORT UNIQUE             |               |  999 |  3996 |          |
|  9 |       INDEX FAST FULL SCAN   | IX_SERV_INST  |  999 |  3996 |    3  (0)|
-------------------------------------------------------------------------------
```

8. Perform a second full scan of the PK_SERVERS index.
9. The cost had been 6 we just added one with the unnecessary duplicate index read

A More Complex Explain Plan (6:8)

```
-------------------------------------------------------------------------------
| Id | Operation                    | Name          | Rows | Bytes |Cost(%CPU)|
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT             |               |    1 |    17 |    9 (45)|
|  1 |  HASH UNIQUE                 |               |    1 |    17 |    9 (45)|
|* 2 |   HASH JOIN ANTI             |               |  140 |  2380 |    8 (38)|
|  3 |    INDEX FULL SCAN           | PK_SERVERS    |  141 |   564 |    1  (0)|
|  4 |    VIEW                      | VW_NSO_1      |  141 |  1833 |    6 (34)|
|  5 |     MINUS                    |               |      |       |          |
|  6 |      SORT UNIQUE             |               |  141 |   564 |          |
|  7 |       INDEX FULL SCAN        | PK_SERVERS    |  141 |   564 |    1  (0)|
|  8 |      SORT UNIQUE             |               |  999 |  3996 |          |
|  9 |       INDEX FAST FULL SCAN   | IX_SERV_INST  |  999 |  3996 |    3  (0)|
-------------------------------------------------------------------------------
```

10. Join the results in the view with the results of the index read.
11. The cost has gone from 7 to 8 of which 38%, or 3, is CPU.

A More Complex Explain Plan (7:8)

```
-------------------------------------------------------------------------------
| Id | Operation                    | Name          | Rows | Bytes |Cost(%CPU)|
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT             |               |    1 |    17 |    9 (45)|
|  1 |  HASH UNIQUE                 |               |    1 |    17 |    9 (45)|
|* 2 |   HASH JOIN ANTI             |               |  140 |  2380 |    8 (38)|
|  3 |    INDEX FULL SCAN           | PK_SERVERS    |  141 |   564 |    1  (0)|
|  4 |    VIEW                      | VW_NSO_1      |  141 |  1833 |    6 (34)|
|  5 |     MINUS                    |               |      |       |          |
|  6 |      SORT UNIQUE             |               |  141 |   564 |          |
|  7 |       INDEX FULL SCAN        | PK_SERVERS    |  141 |   564 |    1  (0)|
|  8 |      SORT UNIQUE             |               |  999 |  3996 |          |
|  9 |       INDEX FAST FULL SCAN   | IX_SERV_INST  |  999 |  3996 |    3  (0)|
-------------------------------------------------------------------------------
```

12. Use a hashing algorithm to collect a set of unique values for the result set
13. The cost has gone from 8 to 9 of which 45%, or 4, is CPU

A More Complex Explain Plan (8:8)

```
-------------------------------------------------------------------------------
| Id | Operation                    | Name          | Rows | Bytes |Cost(%CPU)|
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT             |               |    1 |    17 |    9 (45)|
|  1 |  HASH UNIQUE                 |               |    1 |    17 |    9 (45)|
|* 2 |   HASH JOIN ANTI             |               |  140 |  2380 |    8 (38)|
|  3 |    INDEX FULL SCAN           | PK_SERVERS    |  141 |   564 |    1  (0)|
|  4 |    VIEW                      | VW_NSO_1      |  141 |  1833 |    6 (34)|
|  5 |     MINUS                    |               |      |       |          |
|  6 |      SORT UNIQUE             |               |  141 |   564 |          |
|  7 |       INDEX FULL SCAN        | PK_SERVERS    |  141 |   564 |    1  (0)|
|  8 |      SORT UNIQUE             |               |  999 |  3996 |          |
|  9 |       INDEX FAST FULL SCAN   | IX_SERV_INST  |  999 |  3996 |    3  (0)|
-------------------------------------------------------------------------------
```

14. The result returned to the end-user will be 1 row (17 bytes)

Explain Plans: Bitmap Indexes

```
EXPLAIN PLAN FOR
SELECT *
FROM serv_inst
WHERE location_code = 30386
OR ws_id BETWEEN 326 AND 333;


-----------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name                          | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                               |     2 |   148 |     3   (0)| 00:00:01 |
|   1 |  CONCATENATION                |                               |       |       |            |          |
|   2 |   TABLE ACCESS BY INDEX ROWID | SERV_INST                     |     1 |    74 |     1   (0)| 00:00:01 |
|   3 |    BITMAP CONVERSION TO ROWIDS|                               |       |       |            |          |
|*  4 |     BITMAP INDEX RANGE SCAN   | BIX_SERV_INST_WS_ID           |       |       |            |          |
|*  5 |   TABLE ACCESS BY INDEX ROWID | SERV_INST                     |     1 |    74 |     1   (0)| 00:00:01 |
|   6 |    BITMAP CONVERSION TO ROWIDS|                               |       |       |            |          |
|*  7 |     BITMAP INDEX SINGLE VALUE | BIX_SERV_INST_LOCATION_CODE   |       |       |            |          |
-----------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("WS_ID">=326 AND "WS_ID"<=333)
   5 - filter(LNNVL("WS_ID">=326) OR LNNVL("WS_ID"<=333))
   7 - access("LOCATION_CODE"=30386)
```

Explain Plans: Join Syntax

```
explain plan for
select distinct i.srvr_id
from servers s, serv_inst i
where s.srvr_id = i.srvr_id;
-------------------------------------------------------------------------------------
| Id  | Operation           | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |            |   999 | 25974 |     9  (12)| 00:00:01 |
|   1 |  HASH UNIQUE        |            |   999 | 25974 |     9  (12)| 00:00:01 |
|   2 |   NESTED LOOPS      |            |   999 | 25974 |     8   (0)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| SERV_INST  |   999 | 12987 |     8   (0)| 00:00:01 |
|*  4 |    INDEX UNIQUE SCAN| PK_SERVERS |     1 |    13 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------



explain plan for
select distinct i.srvr_id
from servers s inner join serv_inst i
on s.srvr_id = i.srvr_id;
-------------------------------------------------------------------------------------
| Id  | Operation           | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |            |   999 | 25974 |     9  (12)| 00:00:01 |
|   1 |  HASH UNIQUE        |            |   999 | 25974 |     9  (12)| 00:00:01 |
|   2 |   NESTED LOOPS      |            |   999 | 25974 |     8   (0)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| SERV_INST  |   999 | 12987 |     8   (0)| 00:00:01 |
|*  4 |    INDEX UNIQUE SCAN| PK_SERVERS |     1 |    13 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------
```

Explain Plans: Missing Joins Are Expensive

```
SQL> explain plan for
  2  select s.srvr_id
  3  from servers s, serv_inst i
  4  where s.srvr_id = i.srvr_id;

SQL> select * from table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------
| Id  | Operation          | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |            |   999 | 25974 |     8   (0)| 00:00:01 |
|   1 |  NESTED LOOPS      |            |   999 | 25974 |     8   (0)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| SERV_INST  |   999 | 12987 |     8   (0)| 00:00:01 |
|*  3 |   INDEX UNIQUE SCAN| PK_SERVERS |     1 |    13 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------

SQL> explain plan for
  2  select s.srvr_id
  3  from servers s, serv_inst i;

SQL> select * from table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------
| Id  | Operation                     | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                     |  140K|  1788K|   130   (1)| 00:00:02 |
|   1 |  MERGE JOIN CARTESIAN         |                     |  140K|  1788K|   130   (1)| 00:00:02 |
|   2 |   INDEX FAST FULL SCAN        | PK_SERVERS          |   141 |  1833 |     2   (0)| 00:00:01 |
|   3 |   BUFFER SORT                 |                     |   999 |       |   128   (1)| 00:00:02 |
|   4 |    BITMAP CONVERSION TO ROWIDS |                     |   999 |       |     1   (0)| 00:00:01 |
|   5 |     BITMAP INDEX FAST FULL SCAN| BIX_SERV_INST_WS_ID |       |       |            |          |
-----------------------------------------------------------------------------------------------
```

Explain Plans: Unnecessary Joins Are Expensive

```
SQL> explain plan for
  2  select distinct s.srvr_id
  3  from servers s, serv_inst i
  4  where s.srvr_id = i.srvr_id;

SQL> select * from table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------
| Id  | Operation           | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |            |   999 | 25974 |     9  (12)| 00:00:01 |
|   1 |  HASH UNIQUE        |            |   999 | 25974 |     9  (12)| 00:00:01 |
|   2 |   NESTED LOOPS      |            |   999 | 25974 |     8   (0)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| SERV_INST  |   999 | 12987 |     8   (0)| 00:00:01 |
|*  4 |    INDEX UNIQUE SCAN| PK_SERVERS |     1 |    13 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------


explain plan for
select distinct s1.srvr_id
from servers s1, servers s2, serv_inst i
where s1.srvr_id = s2.srvr_id
and s1.srvr_id = i.srvr_id;
-------------------------------------------------------------------------------
| Id  | Operation           | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |            |   999 | 38961 |    10  (20)| 00:00:01 |
|   1 |  HASH UNIQUE        |            |   999 | 38961 |    10  (20)| 00:00:01 |
|   2 |   NESTED LOOPS      |            |   999 | 38961 |     9  (12)| 00:00:01 |
|   3 |    NESTED LOOPS     |            |   999 | 25974 |     8   (0)| 00:00:01 |
|   4 |     TABLE ACCESS FULL| SERV_INST  |   999 | 12987 |     8   (0)| 00:00:01 |
|*  5 |     INDEX UNIQUE SCAN| PK_SERVERS |     1 |    13 |     0   (0)| 00:00:01 |
|*  6 |    INDEX UNIQUE SCAN | PK_SERVERS |     1 |    13 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

Explain Plans: Parallel Transactions (PX)

```
SQL> EXPLAIN PLAN FOR
  2  SELECT SUM(salary)
  3  FROM emp2
  4  GROUP BY department_id;

Explained.

SQL> SELECT plan_table_output FROM table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------------------------------
Plan hash value: 3939201228
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                | Name     | Rows  | Bytes | Cost (%CPU)| Time     |    TQ  |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------------------------------------------|
|   0 | SELECT STATEMENT         |          |   107 |  2782 |     3  (34)| 00:00:01 |        |      |            |
|   1 |  PX COORDINATOR          |          |       |       |            |          |        |      |            |
|   2 |   PX SEND QC (RANDOM)    | :TQ10001 |   107 |  2782 |     3  (34)| 00:00:01 |  Q1,01 | P->S | QC (       |
|   3 |    HASH GROUP BY         |          |   107 |  2782 |     3  (34)| 00:00:01 |  Q1,01 | PCWP |            |
|   4 |     PX RECEIVE           |          |   107 |  2782 |     3  (34)| 00:00:01 |  Q1,01 | PCWP |            |
|   5 |      PX SEND HASH        | :TQ10000 |   107 |  2782 |     3  (34)| 00:00:01 |  Q1,00 | P->P | HASH       |
|   6 |       HASH GROUP BY      |          |   107 |  2782 |     3  (34)| 00:00:01 |  Q1,00 | PCWP |            |
|   7 |        PX BLOCK ITERATOR |          |   107 |  2782 |     2   (0)| 00:00:01 |  Q1,00 | PCWC |            |
|   8 |         TABLE ACCESS FULL| EMP2     |   107 |  2782 |     2   (0)| 00:00:01 |  Q1,00 | PCWP |            |
-------------------------------------------------------------------------------------------------------------------
Note
-----
   - dynamic sampling used for this statement
```

Explain Plans: Pstart -Pstop: Starting & Stopping Partitions

```
explain plan for
select * from part_zip where state = 'CA';
-------------------------------------------------------------------------------------------------
| Id  | Operation             | Name     | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |          |     3 |    72 |     2   (0)| 00:00:01 |       |       |
|   1 |  PARTITION HASH SINGLE|          |     3 |    72 |     2   (0)| 00:00:01 |     1 |     1 |
|*  2 |   TABLE ACCESS FULL   | PART_ZIP |     3 |    72 |     2   (0)| 00:00:01 |     1 |     1 |
-------------------------------------------------------------------------------------------------


explain plan for
select * from part_zip where state = 'NY';
-------------------------------------------------------------------------------------------------
| Id  | Operation             | Name     | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |          |     3 |    72 |     2   (0)| 00:00:01 |       |       |
|   1 |  PARTITION HASH SINGLE|          |     3 |    72 |     2   (0)| 00:00:01 |     2 |     2 |
|*  2 |   TABLE ACCESS FULL   | PART_ZIP |     3 |    72 |     2   (0)| 00:00:01 |     2 |     2 |
-------------------------------------------------------------------------------------------------


explain plan for
select * from part_zip where zipcode LIKE '%5%';
-------------------------------------------------------------------------------------------------
| Id  | Operation             | Name     | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |          |     2 |    48 |     2   (0)| 00:00:01 |       |       |
|   1 |  PARTITION HASH ALL   |          |     2 |    48 |     2   (0)| 00:00:01 |     1 |     3 |
|*  2 |   TABLE ACCESS FULL   | PART_ZIP |     2 |    48 |     2   (0)| 00:00:01 |     1 |     3 |
-------------------------------------------------------------------------------------------------
```

Explain Plans: Temp Tablespace Usage (ORDER BY clause)

```
SQL> SELECT * FROM TABLE(dbms_xplan.display);

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------
Plan hash value: 995087943

------------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         | 98128 |    12M|       |  3435   (1)| 00:00:42 |
|   1 |  SORT ORDER BY     |         | 98128 |    12M|    25M|  3435   (1)| 00:00:42 |
|   2 |   TABLE ACCESS FULL| SOURCE$ | 98128 |    12M|       |       (2)| 00:00:07 |
------------------------------------------------------------------------------------
```

Explain Plan Errors (1:8)

- Oracle can do math
- But not always as well as you can
- Consider the following SQL statement

```
EXPLAIN PLAN FOR
SELECT srvr_id
FROM servers
INTERSECT
SELECT srvr_id
FROM serv_inst;
```

Explain Plan Errors (2:8)

- Why is this wrong?

```
EXPLAIN PLAN FOR
SELECT srvr_id
FROM servers
INTERSECT
SELECT srvr_id
FROM serv_inst;

 ------------------------------------------------------------------------------
 | Id | Operation                | Name          | Rows | Bytes | Cost (%CPU)|
 ------------------------------------------------------------------------------
 |  0 | SELECT STATEMENT         |               |  141 | 4560  |     6  (84)|
 |  1 |  INTERSECTION            |               |      |       |            |
 |  2 |   SORT UNIQUE NOSORT     |               |  141 |  564  |     2  (50)|
 |  3 |    INDEX FULL SCAN       | PK_SERVERS    |  141 |  564  |     1   (0)|
 |  4 |   SORT UNIQUE            |               |  999 | 3996  |     4  (25)|
 |  5 |    INDEX FAST FULL SCAN| IX_SERV_INST |  999 | 3996  |     3   (0)|
 ------------------------------------------------------------------------------
```

Explain Plan Errors (3:8)

- Why is this wrong?

```
EXPLAIN PLAN FOR
SELECT srvr_id
FROM servers
INTERSECT
SELECT srvr_id
FROM serv_inst;

 ------------------------------------------------------------------------------
 | Id | Operation                | Name          | Rows | Bytes | Cost (%CPU)|
 ------------------------------------------------------------------------------
 |  0 | SELECT STATEMENT         |               |  141 | 4560  |     6  (84)|
 |  1 |  INTERSECTION            |               |      |       |            |
 |  2 |   SORT UNIQUE NOSORT     |               |  141 |  564  |     2  (50)|
 |  3 |    INDEX FULL SCAN       | PK_SERVERS    |  141 |  564  |     1   (0)|
 |  4 |   SORT UNIQUE            |               |  999 | 3996  |     4  (25)|
 |  5 |    INDEX FAST FULL SCAN| IX_SERV_INST |  999 | 3996  |     3   (0)|
 ------------------------------------------------------------------------------
```

1. Read 999 rows, ~4K from the SERV_INST table's index IX_SERV_INST: The cost is 3

Explain Plan Errors (4:8)

- Why is this wrong?

```
EXPLAIN PLAN FOR
SELECT srvr_id
FROM servers
INTERSECT
SELECT srvr_id
FROM serv_inst;

 ------------------------------------------------------------------------------
 | Id | Operation                | Name          | Rows | Bytes | Cost (%CPU)|
 ------------------------------------------------------------------------------
 |  0 | SELECT STATEMENT         |               |  141 | 4560  |     6  (84)|
 |  1 |  INTERSECTION            |               |      |       |            |
 |  2 |   SORT UNIQUE NOSORT     |               |  141 |  564  |     2  (50)|
 |  3 |    INDEX FULL SCAN       | PK_SERVERS    |  141 |  564  |     1   (0)|
 |  4 |   SORT UNIQUE            |               |  999 | 3996  |     4  (25)|
 |  5 |    INDEX FAST FULL SCAN| IX_SERV_INST  |  999 | 3996  |     3   (0)|
 ------------------------------------------------------------------------------
```

2. Sort the IX_SERV_INST index entries
3. The additional cost is 1 (3+1=4) and 25% of the cost of 4 is CPU (4 x 0.25 = 1): The math works

Explain Plan Errors (5:8)

- Why is this wrong?

```
EXPLAIN PLAN FOR
SELECT srvr_id
FROM servers
INTERSECT
SELECT srvr_id
FROM serv_inst;

 ------------------------------------------------------------------------------
 | Id | Operation                | Name          | Rows | Bytes | Cost (%CPU)|
 ------------------------------------------------------------------------------
 |  0 | SELECT STATEMENT         |               |  141 | 4560  |     6  (84)|
 |  1 |  INTERSECTION            |               |      |       |            |
 |  2 |   SORT UNIQUE NOSORT     |               |  141 |  564  |     2  (50)|
 |  3 |    INDEX FULL SCAN       | PK_SERVERS    |  141 |  564  |     1   (0)|
 |  4 |   SORT UNIQUE            |               |  999 | 3996  |     4  (25)|
 |  5 |    INDEX FAST FULL SCAN| IX_SERV_INST |  999 | 3996  |     3   (0)|
 ------------------------------------------------------------------------------
```

4. Read 141 rows, 0.5K, from the primary key of the SERVERS table: The cost is 1
5. This line is indented so it is not added, directly, to the cost of operations 4 and 5

Explain Plan Errors (6:8)

- Why is this wrong?

```
EXPLAIN PLAN FOR
SELECT srvr_id
FROM servers
INTERSECT
SELECT srvr_id
FROM serv_inst;

 ------------------------------------------------------------------------------
 | Id | Operation                | Name          | Rows | Bytes | Cost (%CPU)|
 ------------------------------------------------------------------------------
 |  0 | SELECT STATEMENT         |               |  141 | 4560  |     6  (84)|
 |  1 |  INTERSECTION            |               |      |       |            |
 |  2 |   SORT UNIQUE NOSORT     |               |  141 |  564  |     2  (50)|
 |  3 |    INDEX FULL SCAN       | PK_SERVERS    |  141 |  564  |     1   (0)|
 |  4 |   SORT UNIQUE            |               |  999 | 3996  |     4  (25)|
 |  5 |    INDEX FAST FULL SCAN  | IX_SERV_INST  |  999 | 3996  |     3   (0)|
 ------------------------------------------------------------------------------
```

6. A SORT UNIQUE NOSORT is used to remove potential duplicate rows
7. The additional cost is 1 (1+1=2) and 50% of the cost of 2 is CPU (2 x 0.50 = 1): The math works again

Explain Plan Errors (7:8)

- Why is this wrong?

```
EXPLAIN PLAN FOR
SELECT srvr_id
FROM servers
INTERSECT
SELECT srvr_id
FROM serv_inst;

 ------------------------------------------------------------------------------
 | Id | Operation                | Name          | Rows | Bytes | Cost (%CPU)|
 ------------------------------------------------------------------------------
 |  0 | SELECT STATEMENT         |               |  141 | 4560  |     6  (84)|
 |  1 |  INTERSECTION            |               |      |       |            |
 |  2 |   SORT UNIQUE NOSORT     |               |  141 |  564  |     2  (50)|
 |  3 |    INDEX FULL SCAN       | PK_SERVERS    |  141 |  564  |     1   (0)|
 |  4 |   SORT UNIQUE            |               |  999 | 3996  |     4  (25)|
 |  5 |    INDEX FAST FULL SCAN| IX_SERV_INST |  999 | 3996  |     3   (0)|
 ------------------------------------------------------------------------------
```

8. Perform an intersection of the two result sets

Explain Plan Errors (8:8)

- Why is this wrong?

```
EXPLAIN PLAN FOR
SELECT srvr_id
FROM servers
INTERSECT
SELECT srvr_id
FROM serv_inst;

 ------------------------------------------------------------------------------
 | Id | Operation                | Name          | Rows | Bytes | Cost (%CPU)|
 ------------------------------------------------------------------------------
 |  0 | SELECT STATEMENT         |               |  141 | 4560  |     6  (84)|
 |  1 |  INTERSECTION            |               |      |       |            |
 |  2 |   SORT UNIQUE NOSORT     |               |  141 |  564  |     2  (50)|
 |  3 |    INDEX FULL SCAN       | PK_SERVERS    |  141 |  564  |     1   (0)|
 |  4 |   SORT UNIQUE            |               |  999 | 3996  |     4  (25)|
 |  5 |    INDEX FAST FULL SCAN  | IX_SERV_INST  |  999 | 3996  |     3   (0)|
 ------------------------------------------------------------------------------
```

9. Add the two costs (2+4=6): The math works
10. 25% of the 4 is CPU (1) and 50% of the 2 is CPU (1) and (1+1=2). Is 2/6 equal to 84%?

Explain Planning a Statement After It Has Been Run

- Use the DISPLAY_CURSOR pipelined table function

```
dbms_xplan.display_cursor(
sql_id          IN VARCHAR2 DEFAULT NULL,
cursor_child_no IN INTEGER DEFAULT 0,
format          IN VARCHAR2 DEFAULT 'TYPICAL')
RETURN dbms_xplan_type_table PIPELINED;
```

Format Constants	
ALIAS	If relevant, shows the "Query Block Name / Object Alias" section
ALLSTATS	A shortcut for 'IOSTATS MEMSTATS'
BYTES	If relevant, shows the number of bytes estimated by the optimizer
COST	If relevant, shows optimizer cost information
IOSTATS	Assuming that basic plan statistics are collected when SQL statements are executed (either by using the gather_plan_statistics hint or by setting the parameter statistics_level to ALL), this format will show IO statistics for ALL (or only for the LAST as shown below) executions of the cursor
LAST	By default, plan statistics are shown for all executions of the cursor. The keyword LAST can be specified to see only the statistics for the last execution
MEMSTATS	Assuming that PGA memory management is enabled (that is, pga_aggregate_target parameter is set to a non 0 value), this format allows to display memory management statistics (for example, execution mode of the operator, how much memory was used, number of bytes spilled to disk, and so on). These statistics only apply to memory intensive operations like hash-joins, sort or some bitmap operators
NOTE	If relevant, shows the note section of the explain plan
PARALLEL	If relevant, shows PX information (distribution method and table queue information)
PARTITION	If relevant, shows partition pruning information
PREDICATE	If relevant, shows the predicate section
PROJECTION	If relevant, shows the projection section
REMOTE	If relevant, shows the information for distributed query (for example, remote from serial distribution and remote SQL)
ROWS	If relevant, shows the number of rows estimated by the optimizer
RUNSTATS_LAST	Same as IOSTATS LAST: displays the runtime stat for the last execution of the cursor
RUNSTATS_TOT	Same as IOSTATS: displays IO statistics for all executions of the specified cursor

Explain Planning a Statement After It Has Been Run

- Use the DISPLAY_CURSOR pipelined table function

```
SELECT /* XPLAN_CURSOR */ DISTINCT s.srvr_id
FROM servers s, serv_inst I
WHERE s.srvr_id = i.srvr_id;

SELECT sql_id
FROM gv$sql
WHERE sql_text LIKE '%XPLAN_CURSOR%';

SELECT * FROM
TABLE(dbms_xplan.display_cursor('cpm9ss48qd32f'));
```

Explain Planning a Statement After It Has Been Run

First get the SQL_ID of the statement

```
SQL> SELECT DISTINCT sql_id
  2  FROM v$sqlarea
  3  WHERE executions = (SELECT
MAX(executions) FROM v$sqlarea);

SQL_ID
-------------
5sg7mjrpj21z7
fnq8p3fj3r5as
```

Explain Planning a Statement After Execution

```
SQL> SELECT * FROM TABLE(dbms_xplan.display_cursor('5sg7mjrpj21z7'));

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------
SQL_ID  5sg7mjrpj21z7, child number 0
-------------------------------------
SELECT JOB,LAST_DATE,THIS_DATE,NEXT_DATE,FIELD1 FROM SYS."JOB$_REDUCED"
"JOB$_REDUCED" WHERE "JOB$_REDUCED"."THIS_DATE" IS NULL AND
("JOB$_REDUCED"."LAST_DATE" IS NULL AND "JOB$_REDUCED"."NEXT_DATE"<:1
OR "JOB$_REDUCED"."NEXT_DATE">=:2 AND "JOB$_REDUCED"."NEXT_DATE"<=:3)
AND ("JOB$_REDUCED"."FIELD1"=:4 OR "JOB$_REDUCED"."FIELD1"=0 AND
'Y'=:5) AND ("JOB$_REDUCED"."JOB"<1000000000 AND
"SYS"."DBMS_LOGSTDBY"."DB_IS_LOGSTDBY"()=0 OR "JOB$_REDUCED"."JOB">=1000000000 AND
"SYS"."DBMS_LOGSTDBY"."DB_IS_LOGSTDBY"()=1)

Plan hash value: 3312758264
------------------------------------------------------------------------------------
| Id  | Operation                                | Name        | Rows  | Bytes | Cost (%CPU)|
------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |             |       |       |     1 (100)|
|   1 |  CONCATENATION                           |             |       |       |            |
|*  2 |   TABLE ACCESS BY INDEX ROWID BATCHED| JOB$         |     1 |    53 |     0   (0)|
|*  3 |    INDEX RANGE SCAN                      | I_JOB_NEXT  |     1 |       |     0   (0)|
|*  4 |   TABLE ACCESS BY INDEX ROWID BATCHED| JOB$         |     1 |    53 |     0   (0)|
|*  5 |    INDEX RANGE SCAN                      | I_JOB_NEXT  |     1 |       |     0   (0)|
------------------------------------------------------------------------------------
```

Related Dynamic Performance View

- The following dynamic performance views give information vital to understanding how SQL is running in the real-world environment
  - V$SQL
  - V$SQLAREA
  - V$SQLSTATS
  - V$SQL_BIND_CAPTURE
  - V$SQL_BIND_DATA
  - V$SQL_BIND_METADATA
  - V$SQL_PLAN

One Way To Look At Performance

```
SQL> EXPLAIN PLAN FOR
  2  SELECT COUNT(*)
  3  FROM parent p, child c
  4  WHERE p.parent_id = c.parent_id;

SQL> select * From table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 3584092213
--------------------------------------------------------------------------------
| Id  | Operation           | Name   | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |        |     1 |    10 |       |  3163   (5)| 00:00:38 |
|   1 |  SORT AGGREGATE     |        |     1 |    10 |       |            |          |
|*  2 |   HASH JOIN         |        |  1500K|    14M|  8312K|  3163   (5)| 00:00:38 |
|   3 |    TABLE ACCESS FULL| PARENT |   500K|  2442K|       |   380   (4)| 00:00:05 |
|   4 |    TABLE ACCESS FULL| CHILD  |  1500K|  7324K|       |  1106   (5)| 00:00:14 |
--------------------------------------------------------------------------------


SQL> explain plan for
  2  SELECT COUNT(*)
  3  FROM parent p, child c
  4  WHERE p.parent_id = c.parent_id
  5  AND c.birth_date is NOT NULL;

SQL> select * From table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 3584092213
--------------------------------------------------------------------------------
| Id  | Operation           | Name   | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |        |     1 |    16 |       |  3037   (5)| 00:00:37 |
|   1 |  SORT AGGREGATE     |        |     1 |    16 |       |            |          |
|*  2 |   HASH JOIN         |        |   999K|    15M|  8312K|  3037   (5)| 00:00:37 |
|   3 |    TABLE ACCESS FULL| PARENT |   500K|  2442K|       |   380   (4)| 00:00:05 |
|*  4 |    TABLE ACCESS FULL| CHILD  |   999K|    10M|       |  1116   (6)| 00:00:14 |
--------------------------------------------------------------------------------
```

Another Way To Look At Performance

```
SQL> set timing on

SQL> SELECT COUNT(*)
  2  FROM parent p, child c
  3  WHERE p.parent_id = c.parent_id;

  COUNT(*)
----------
   1500000

Elapsed: 00:00:00.59



SQL> SELECT COUNT(*)
  2  FROM parent p, child c
  3  WHERE p.parent_id = c.parent_id
  4  AND birth_date is NOT NULL;

  COUNT(*)
----------
   1000000

Elapsed: 00:00:00.53
```

Adaptive Execution Plans (1:2)

- New to Database version 12c is Adaptive Execution Plans and a newly tracked feature is use of "adaptive execution plans"
- An adaptive plan is one that learns about the data as it is executed

```
DECLARE
 i NUMBER;
 j NUMBER;
 k CLOB;
BEGIN
  dbms_feature_adaptive_plans(i, j, k);
  dbms_output.put_line('1: ' || i);
  dbms_output.put_line('2: ' || j);
  dbms_output.put_line('3: ' || k);
END;
/
1: 1
2:
3: Total number of queries: 501
Number of queries with an adaptive plan: 35
Percentage of queries with an adaptive plan:
6.986027944111776447105788423153692614 77
Are the queries running in reporting mode ? : No
```

Adaptive Execution Plans (2:2)

- An option available to the DBMS_XPLAN built-in package is through the use of the format constant ADAPTIVE which
  - Displays the final plan, or the current plan if the execution has not completed
    - This section includes notes about runtime optimizations that affect the plan, such as switching from a Nested Loops join to a Hash join
  - Plan lineage
    - This section shows the plans that were run previously due to automatic reoptimization
    - It also shows the default plan, if the plan changed due to dynamic plans
  - Recommended plan
    - In reporting mode, the plan is chosen based on execution statistics displayed
    - Note that displaying the recommended plan for automatic reoptimization requires re-compiling the query with the optimizer adjustments collected in the child cursor
    - Displaying the recommended plan for a dynamic plan does not require this
  - Dynamic plans
    - This summarizes the portions of the plan that differ from the default plan chosen by the optimizer

This Makes It Even Worse

```
SQL> DECLARE
  2   uf   NUMBER;
  3   ub   NUMBER;
  4   f1   NUMBER;
  5   f1b  NUMBER;
  6   f2   NUMBER;
  7   f2b  NUMBER;
  8   f3   NUMBER;
  9   f3b  NUMBER;
 10   f4   NUMBER;
 11   f4b  NUMBER;
 12   fbl  NUMBER;
 13   fby  NUMBER;
 14  BEGIN
 15    dbms_space.space_usage('UWCLASS','SERVERS', 'TABLE', uf, ub, f1, f1b, f2, f2b, f3, f3b, f4, f4b, fbl, fby);
 16
 17    dbms_output.put_line('unformatted blocks:  ' || TO_CHAR(uf));
 18    dbms_output.put_line('unformatted bytes:   ' || TO_CHAR(ub));
 19    dbms_output.put_line('blocks 0-25% free:   ' || TO_CHAR(f1));
 20    dbms_output.put_line('bytes 0-25% free:    ' || TO_CHAR(f1b));
 21    dbms_output.put_line('blocks 25-50% free:  ' || TO_CHAR(f2));
 22    dbms_output.put_line('bytes 25-50% free:   ' || TO_CHAR(f2b));
 23    dbms_output.put_line('blocks 50-75% free:  ' || TO_CHAR(f3));
 24    dbms_output.put_line('bytes 50-75% free:   ' || TO_CHAR(f3b));
 25    dbms_output.put_line('blocks 75-100% free: ' || TO_CHAR(f4));
 26    dbms_output.put_line('bytes 75-100% free:  ' || TO_CHAR(f4b));
 27    dbms_output.put_line('full blocks:         ' || TO_CHAR(fbl));
 28    dbms_output.put_line('full bytes:          ' || TO_CHAR(fby));
 29  END;
 30  /
unformatted blocks:  16
unformatted bytes:   131072
blocks 0-25% free:   0
bytes 0-25% free:    0
blocks 25-50% free:  1
bytes 25-50% free:   8192
blocks 50-75% free:  0
bytes 50-75% free:   0
blocks 75-100% free: 11
bytes 75-100% free:  90112
full blocks:         0
full bytes:          0
```

only 12 blocks are formatted

more examples: www.morganslibrary.org/reference/pkgs/dbms_space.html

AWR Reports: Used and Abused

Foreground Wait Events

- s - second, ms - millisecond - 1000th of a second
- Only events with Total Wait Time (s) >= .001 are shown
- ordered by wait time desc, waits desc (idle events last)
- %Timeouts: value of 0 indicates value was < .5%. Value of null is truly 0

Event	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn	% DB time
reliable message	6,594,174	44	3,293,838	500	0.96	9.88
enq: TX - row lock contention	21,667	15	3,150,728	145416	0.00	9.45
library cache pin	3,784,534	0	2,624,309	693	0.55	7.87
DFS lock handle	2,598,761	11	2,480,524	955	0.38	7.44
db file sequential read	719,646,074	0	2,364,757	3	104.53	7.09
enq: PS - contention	6,147,054	75	1,819,338	296	0.89	5.46
library cache lock	2,737,270	1	1,335,137	488	0.40	4.00
row cache lock	2,437,346	10	1,319,231	541	0.35	3.96
enq: FB - contention	683,161	0	651,060	953	0.10	1.95
PX Deq: Slave Session Stats	4,204,746	22	585,857	139	0.61	1.76
unspecified wait event	763,681,122	0	427,377	1	110.93	1.28
rdbms ipc reply	696,586	20	374,694	538	0.10	1.12
db file scattered read	11,920,396	0	259,432	22	1.73	0.78
direct path read temp	45,839,102	0	200,752	4	6.66	0.60
enq: HW - contention	189,193	0	194,056	1026	0.03	0.58
enq: TM - contention	416,868	92	170,082	408	0.06	0.51
gc cr grant 2-way	113,352,735	0	125,939	1	16.46	0.38
ksim generic wait event	488,898	99	97,114	199	0.07	0.29
enq: JS - queue lock	5,977	0	89,887	15039	0.00	0.27
gcs drm freeze in enter server mode	198,923	78	87,797	441	0.03	0.26
resmgr:cpu quantum	6,065,202	0	87,304	14	0.88	0.26
latch free	2,863,085	0	83,028	29	0.42	0.25
enq: TT - contention	46,944	0	72,635	1547	0.01	0.22
read by other session	15,427,888	0	69,678	5	2.24	0.21

AWR Difference Reports (1:5)

- SQL scripts located at $ORACLE_HOME/rdbms/admin

File Name	Strategy
awrddrpi.sql	Report on differences between differences between values recorded in two pairs of snapshots. This script requests the user for the dbid and instance number of the instance to report on, for each snapshot pair, before producing the standard report.
awrddrpt.sql	Defaults the dbid and instance number to that of the current instance connected-to, then calls awrddrpi.sql to produce the Compare Periods report.
awrextr.sql	SQL/Plus script to help users extract data from the AWR.
awrgdrpi.sql	RAC Version of Compare Period Report.
awrgdrpt.sql	This script defaults the dbid to that of the current instance connected-to, defaults instance list to all available instances and then calls awrgdrpi.sql to produce the Workload Repository RAC Compare Periods report.
awrgrpt.sql	This script defaults the dbid to that of the current instance connected-to, then calls awrgrpti.sql to produce the Workload Repository RAC report.
awrgrti.sql	SQL*Plus command file to report on RAC-wide differences between values recorded in two snapshots. This script requests the user for the dbid before producing the standard Workload Repository report.

more examples: www.morganslibrary.org/reference/awr_report.html

AWR Difference Reports (2:5)

- SQL scripts located at $ORACLE_HOME/rdbms/admin

File Name	Strategy
awrinfo.sql	Outputs general AWR information such as the size, data distribution, etc. in AWR and SYSAUX. The intended use of this script is for diagnosing abnormalities in AWR and not for diagnosing issues in the database instance.
awrrpt.sql	Defaults the dbid and instance number to that of the current instance connected-to, then calls awrrpti.sql to produce the report.
awrrpti.sql	SQL*Plus command file to report on differences between values recorded in two snapshots. This script requests the user for the dbid and instance number of the instance to report on, before producing the standard report.
awrsqrpi.sql	SQL*Plus command file to report on differences between values recorded in two snapshots. This script requests the user for the dbid, instance number and the sql id, before producing a report for a particular sql statement in this instance.
awrsqrpt.sql	Defaults the dbid and instance number to that of the current instance connected-to then calls awrsqrpi.sql to produce a Workload report for a particular sql statement.
awrupd12.sql	This script updates AWR data to version 12c. It only modifies AWR data that has been imported using awrload.sql, or data from before changing the database DBID. In other words, it doesn't modify AWR data for the local, active DBID.

more examples: www.morganslibrary.org/reference/awr_report.html

AWR Difference Reports (3:5)

WORKLOAD REPOSITORY COMPARE PERIOD REPORT

Snapshot Set	DB Name	DB Id	Instance	Inst num	Release	Cluster	Host	Std Block Size
First (1st)	ORAP09	2886124853	orap09	1	11.2.0.3.0	NO	db20p03sh	8192
Second (2nd)	ORAP09	2886124853	orap09	1	11.2.0.3.0	NO	db20p03sh	8192

Snapshot Set	Begin Snap Id	Begin Snap Time	End Snap Id	End Snap Time	Avg Active Users	Elapsed Time (min)	DB time (min)
1st	33759	02-Apr-14 04:00:19 (Wed)	33760	02-Apr-14 04:30:22 (Wed)	0.8	30.1	24.5
2nd	33807	03-Apr-14 04:00:13 (Thu)	33808	03-Apr-14 04:30:16 (Thu)	1.0	30.1	29.8
%Diff					*22.2*	*0.0*	*21.6*

Host Configuration Comparison

	1st	2nd	Diff	%Diff
Number of CPUs:	80	80	0	0.0
Number of CPU Cores:	40	40	0	0.0
Number of CPU Sockets:	4	4	0	0.0
Physical Memory:	1031464.9M	1031464.9M	0M	0.0
Load at Start Snapshot:	19.09	20.68	1.59	8.3
Load at End Snapshot:	11.49	11.18	-.31	-2.7
%User Time:	14.88	14.44	-.44	-3.0
%System Time:	1.08	1.05	-.03	-2.8
%Idle Time:	83.92	84.38	.46	0.5
%IO Wait Time:	.31	.45	.13	45.2

more examples: www.morganslibrary.org/reference/awr_report.html

AWR Difference Reports (4:5)

Wait Events

- Ordered by absolute value of 'Diff' column of '% of DB time' descending (idle events last)

		% of DB time			# Waits/sec (Elapsed Time)			Total Wait Time (sec)			Avg Wait Time (ms)		
Event	Wait Class	1st	2nd	Diff	1st	2nd	%Diff	1st	2nd	%Diff	1st	2nd	%Diff
db file scattered read	User I/O	0.01	4.46	4.45	2.30	544.72	23,583.48	0.21	79.69	37,847.62	0.05	0.08	60.00
read by other session	User I/O	0.00	0.88	0.87	0.35	66.08	18,780.00	0.03	15.65	52,066.67	0.05	0.13	160.00
log file parallel write	System I/O	3.80	3.41	-0.39	11.54	11.64	0.87	55.83	60.88	9.05	2.68	2.90	8.21
db file sequential read	User I/O	0.66	0.85	0.19	12.60	108.69	762.62	9.72	15.24	56.79	0.43	0.08	-81.40
enq: CR - block range reuse dkpt	Other	0.26	0.17	-0.09	0.40	0.31	-22.50	3.83	2.99	-21.93	5.24	5.35	2.10
log file sync	Commit	3.17	3.11	-0.06	8.42	9.26	9.98	46.65	55.59	19.16	3.07	3.33	8.47
SQL*Net message to client	Network	0.06	0.12	0.06	724.68	1,868.96	157.90	0.85	2.08	144.71	0.00	0.00	0.00
utl_file I/O	User I/O	0.23	0.19	-0.04	85.50	84.68	-0.96	3.40	3.34	-1.76	0.02	0.02	0.00
enq: RO - fast object reuse	Application	0.12	0.08	-0.03	0.41	0.31	-24.39	1.72	1.50	-12.79	2.34	2.68	14.53
db file async I/O submit	System I/O	1.29	1.26	-0.03	5.49	5.96	8.56	18.97	22.51	18.66	1.91	2.09	9.42
control file parallel write	System I/O	0.35	0.32	-0.03	0.75	0.73	-2.67	5.09	5.66	11.20	3.75	4.32	15.20
library cache: mutex X	Concurrency	0.05	0.02	-0.03	1.46	1.10	-24.66	0.77	0.42	-45.45	0.29	0.21	-27.59
latch free	Other	0.05	0.03	-0.02	0.34	0.31	-8.82	0.76	0.60	-21.05	1.23	1.08	-12.20
direct path write	User I/O	0.06	0.05	-0.01	1.67	2.14	28.14	0.90	0.85	-5.56	0.30	0.22	-26.67
direct path write temp	User I/O	0.37	0.39	0.01	1.52	2.30	51.32	5.46	6.88	26.01	1.99	1.66	-16.58
reliable message	Other	0.03	0.02	-0.01	0.81	0.62	-23.46	0.47	0.37	-21.28	0.32	0.33	3.13
log buffer space	Configuration	0.02	0.01	-0.01	0.01	0.01	0.00	0.23	0.11	-52.17	14.15	8.65	-38.87
db file parallel write	System I/O	0.03	0.02	-0.01	0.10	0.08	-20.00	0.40	0.38	-5.00	2.31	2.71	17.32
control file sequential read	System I/O	0.01	0.00	-0.00	5.45	4.86	-10.83	0.10	0.09	-10.00	0.01	0.01	0.00
SQL*Net more data from client	Network	0.01	0.01	-0.00	1.98	1.89	-4.55	0.11	0.10	-9.09	0.03	0.03	0.00
SQL*Net more data to client	Network	0.01	0.01	-0.00	3.84	3.94	2.60	0.12	0.12	0.00	0.02	0.02	0.00
asynch descriptor resize	Other	0.00	0.00	-0.00	2.45	2.43	-0.82	0.05	0.05	0.00	0.01	0.01	0.00
cursor: pin S	Concurrency	0.00	0.00	0.00	0.04	0.03	-25.00	0.00	0.01	100.00	0.01	0.23	2,200.00
latch: redo writing	Configuration	0.00	0.00	-0.00	0.06	0.04	-33.33	0.02	0.01	-50.00	0.17	0.16	-5.88
latch: messages	Other	0.00	0.00	-0.00	0.16	0.13	-18.75	0.04	0.04	0.00	0.14	0.17	21.43
direct path read temp	User I/O	0.00	0.00	-0.00	0.63	0.48	-23.81	0.02	0.01	-50.00	0.02	0.02	0.00
direct path sync	User I/O	0.01	0.00	-0.00	0.02	0.03	50.00	0.08	0.09	12.50	2.19	1.76	-19.63
latch: cache buffers chains	Concurrency	0.00	0.00	0.00	0.14	0.63	350.00	0.02	0.03	50.00	0.06	0.02	-66.67

more examples: www.morganslibrary.org/reference/awr_report.html

AWR Difference Reports (5:5)

Other Instance Activity Stats

- Ordered by statistic name

Statistic	Value			per Second (DB time)			per Second (Elapsed Time)			per Trans		
	1st	2nd	%Diff	1st	2nd	%Diff	1st	2nd	%Diff	1st	2nd	%Diff
calls to kcmgas	44,985	48,775	8.43	30.62	27.31	-10.81	24.94	27.04	8.42	2.48	2.51	1.21
calls to kcmgcs	327,166	351,412	7.41	222.67	196.76	-11.64	181.40	194.84	7.41	18.03	18.07	0.22
cell physical IO interconnect bytes	2,106,564,608	100,858,843,136	4,687.84	1,433,726.13	56,473,555.08	3,838.94	1,167,996.50	55,921,065.97	4,687.78	116,108.95	5,186,077.91	4,366.56
change write time	518	468	-9.65	0.35	0.26	-25.71	0.29	0.26	-10.34	0.03	0.02	-33.33
cleanout - number of ktugct calls	1,431	1,349	-5.73	0.97	0.76	-21.65	0.79	0.75	-5.06	0.08	0.07	-12.50
cleanouts and rollbacks - consistent read gets	10	29	190.00	0.01	0.02	100.00	0.01	0.02	100.00	0.00	0.00	0.00

more examples: www.morganslibrary.org/reference/awr_report.html

DBMS_ADDM

- COMPARE_DATABASES NEW new in Database 12c
  - Create a report comparing the performance of a database over two different time periods or the performance of two different databases over two different time periods
- COMPARE_INSTANCES NEW new in Database 12c
  - Create a report comparing the performance of a single instance over two different time periods or the performance of two different instances over two different time periods

```
dbms_addm.compar_databases(
 base_dbid           IN NUMBER,
 base_begin_snap_id  IN NUMBER,
 base_end_snap_id    IN NUMBER,
 comp_dbid           IN NUMBER,
 comp_begin_snap_id  IN NUMBER,
 comp_end_snap_id    IN NUMBER,
 report_type         IN VARCHAR2 := 'HTML')
 RETURN CLOB
```

```
dbms_addm.compare_instances(
 base_dbid           IN NUMBER,
 base_instance_id    IN NUMBER,
 base_begin_snap_id  IN NUMBER,
 base_end_snap_id    IN NUMBER,
 comp_dbid           IN NUMBER,
 comp_instance_id    IN NUMBER,
 comp_begin_snap_id  IN NUMBER,
 comp_end_snap_id    IN NUMBER,
 report_type         IN VARCHAR2 := 'HTML')
 RETURN CLOB;
```

Application Design

Too Many Columns (1:2)

- Oracle claims that a table can contain up to 1,000 columns: It is not true. No database can do 1,000 columns no matter what their marketing claims may be
- The maximum number of real table columns is 255
- Break the 255 barrier and optimizations such as advanced and hybrid columnar compression no longer work
- A 1,000 column table is actually four segments joined together seamlessly behind the scenes just as a partitioned table appears to be a single segment but isn't
- Be suspicious of any table with more than 50 columns. At 100 columns it is time to take a break and re-read the Codd-Date rules on normalization
- Think vertically not horizontally

Too Many Columns (2:2)

- Be very suspicious of any table with column names in the form "SPARE1", "SPARE2", "..."
- The more columns a table has the more cpu is required when accessing columns to the right (as the table is displayed in a SELECT * query ... or at the bottom if the table is displayed by a DESCribe)

Column Ordering (1:2)

- Computers are not humans and tables are not paper forms
- CBO's column retrieval cost
  - Oracle stores columns in variable length format
  - Each row is parsed in order to retrieve one or more columns
  - Each subsequently parsed column introduces a cost of 20 cpu cycles regardless of whether it is of value or not
  - These tables will be accessed by person_id or state: No one will ever put the address2 column into the WHERE clause as a filter ... they won't filter on middle initial either

Common Design

```
CREATE TABLE customers (
person_id   NUMBER,
first_name  VARCHAR2(30) NOT NULL,
middle_init VARCHAR2(2),
last_name   VARCHAR2(30) NOT NULL,
address1    VARCHAR2(30),
address2    VARCHAR2(30),
city        VARCHAR2(30),
state       VARCHAR2(2));
```

Optimized Design

```
CREATE TABLE customers (
person_id   NUMBER,
last_name   VARCHAR2(30) NOT NULL,
state       VARCHAR2(2)  NOT NULL,
city        VARCHAR2(30) NOT NULL,
first_name  VARCHAR2(30) NOT NULL,
address1    VARCHAR2(30),
address2    VARCHAR2(30),
middle_init VARCHAR2(2));
```

Column Ordering (2:2)

- Proof column order matters

```
CREATE TABLE read_test AS
SELECT *
FROM apex_040200.wwv_flow_page_plugs
WHERE rownum = 1;

SQL> explain plan for
  2  select * from read_test;

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
| Id  | Operation          | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |     1 |   214K|     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| READ_TEST |     1 |   214K|     2   (0)| 00:00:01 |
------------------------------------------------------------------------------

-- fetch value from column 1
Final cost for query block SEL$1 (#0) - All Rows Plan:
  Best join order: 1
  Cost: 2.0002  Degree: 1  Card: 1.0000  Bytes: 13
  Resc: 2.0002  Resc_io: 2.0000  Resc_cpu: 7271
  Resp: 2.0002  Resp_io: 2.0000  Resc_cpu: 7271

-- fetch value from column 193
Final cost for query block SEL$1 (#0) - All Rows Plan:
  Best join order: 1
  Cost: 2.0003  Degree: 1  Card: 1.0000  Bytes: 2002
  Resc: 2.0003  Resc_io: 2.0000  Resc_cpu: 11111
  Resp: 2.0003  Resp_io: 2.0000  Resc_cpu: 11111
```

More Object Optimizations

Index Mythology

- Full Table Scans are bad
  - The above statement is pure unadulterated nonsense: Easily proven false and yet it persists year-after-year
- Queries should use an index
- There is a magic number, like 10% or rows, over which an index won't be used
- All indexes are of equal value
- Indexes need to be rebuilt regularly
  - Rebuilds are DDL and require substantial resources and locking
  - Indexes almost never need to be rebuilt
  - Coalesced perhaps but not rebuilt

Myth Busted


```
SQL> explain plan for
  2  SELECT doc_name
  3  FROM t
  4  WHERE person_id = 221;

SQL> select * from table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |   216 |  6264 |    64   (4)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| T    |   216 |  6264 |    64   (4)| 00:00:01 |
------------------------------------------------------------------------------
```

```
SQL> explain plan for
  2  SELECT /*+ INDEX(t ix_t_person_id) */ doc_name
  3  FROM t
  4  WHERE person_id = 221;

SQL> select * from table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |                |   216 |  6264 |   216   (0)| 00:00:03 |
|   1 |  TABLE ACCESS BY INDEX ROWID| T              |   216 |  6264 |   216   (0)| 00:00:03 |
|*  2 |   INDEX RANGE SCAN          | IX_T_PERSON_ID |   216 |       |     1   (0)| 00:00:01 |
------------------------------------------------------------------------------------------------
```

Oracle Index Types

- Unlike SQL Server, and other products, Oracle has many types of indexes so your choice of index type can be optimized for the application's needs
  - B*Tree
  - Bitmap
  - Bitmap Join
  - Compressed
  - Descending
  - Function Based
  - Hash
  - IndexType
  - Invisible
  - Reverse
  - XML

B*Tree Indexes (1:2)

- Balance Tree
- Upside down Tree-like structure where each branch represents a different set of data
- Contains branch blocks and leaf blocks
- Leaf blocks contain index values and the ROWID that points to the physical location of a row with that value
- The default index type in Oracle
- The only index type in most RDBMSs

```
SQL> CREATE INDEX ix_postal_codes
  2  ON postal_codes(state, city)
  3  TABLESPACE uwdata;
```

Concatenated Indexes

- With concatenated indexes, the leading column should be the most selective
- If the leading column is the most selective and the most frequently used in the limiting conditions
- Then the second and subsequent columns of the index will have the most effect if they are the next most selective and the next most frequently queried
- This will have the biggest impact on the biggest number of queries, assuming that the same column is used most frequently in the limiting conditions of queries
- It also makes possible BASIC Index compression

Concatenated Indexes vs. Multiple Single Column Indexes

- If the other columns are frequently referenced with the leading column, there will be significant performance advantages by creating a concatenated index instead of separate indexes
- Less I/O will be required
- The results from the separate index reads won't need to be merged, resulting in less filtering

Bitmap and Bitmap Join Indexes

- Use with low cardinality (a small number of distinct values) and essentially no updates or deletes
- The word possible example of low cardinality for a bitmap index is a gender column with the values "F" and "M": Why?
- When you modify one byte in one row of a table with a bitmap index on the changed column how much redo is written? This is critically important to know
- If it isn't a traditional Data Warehouse do not use a Bitmap Index
- If you update a single byte in a B*Tree index how much redo is generated?

Descending Indexes

- Optimal when an index read is associated with an ORDER BY DESCENDING clause

```
SQL> CREATE TABLE orders(
  2  order_id    NUMBER(9,0),
  3  ship_date   DATE;

SQL> CREATE INDEX ix_orders_reg
  2  ON orders(cust_id, ship_date);

SQL> CREATE INDEX ix_orders_desc
  2  ON orders(cust_id, ship_date DESC);

SQL> EXPLAIN PLAN FOR
  2  SELECT order_id
  3  FROM orders
  4  WHERE ship_date IS NOT NULL
  5  ORDER BY ship_date DESC;

-----------------------------------------------
| Id  | Operation             | Name           |
-----------------------------------------------
|   0 | SELECT STATEMENT      |                |
|   1 |  SORT ORDER BY        |                |
|*  2 |   INDEX FAST FULL SCAN| IX_ORDERS_DESC |
-----------------------------------------------
```

Function Based Indexes (1 of 2)

- Index the result of a function applied to a column or multiple columns within the row
- Can be used to enhance performance by not indexing that which is of no value to index ... for example a column with 10% "Y" and 90% "No" values
- Without a Function Based Index a full table scan would be required as well as a calculation of every row

```
conn scott/tiger@pdborcl

SQL> desc emp
Name                 Null?    Type
-------------------- -------- ------------
EMPNO                NOT NULL NUMBER(4)
ENAME                         VARCHAR2(10)
JOB                           VARCHAR2(9)
MGR                           NUMBER(4)
HIREDATE                      DATE
SAL                           NUMBER(7,2)
COMM                          NUMBER(7,2)
DEPTNO                        NUMBER(2)

CREATE INDEX fbi_emp_sal_x_comm
ON emp(sal + comm);

SELECT ename
FROM emp
WHERE (sal + comm) < 300000;
```

Function Based Indexes (2 of 2)

- A regular index would index 99% "N" values wasting substantial space and the index will never be used to find the value "N"
- As a normal B*Tree index IX_FBIDEMO will have what leaf block distribution?

```
SQL> CREATE TABLE fbidemo AS
  2  SELECT object_name, object_type, temporary
  3  FROM dba_objects;

SQL> CREATE INDEX ix_fbidemo
  2  ON fbidemo (temporary);

SQL> CREATE INDEX fbi_fbidemo
  2  ON fbidemo (DECODE(temporary, 'Y', 'Y', NULL));
```

Invisible Indexes

- Invisible indexes are real indexes and are maintained as like any other index but, by default, are not visible to the optimizer
- Excellent for use when inappropriate use of an index might slow down the system but a small number of sessions will benefit from it

```
CREATE INDEX ix_invis
ON invis(table_name)
INVISIBLE;

ALTER SESSION SET optimizer_use_invisible_indexes = TRUE;
```

Sorted Hash Clusters

- Sorted hash clusters, applied appropriately, can eliminate substantial overhead created by an ORDER BY clause

```
CREATE CLUSTER sorted_hc (
program_id  NUMBER(3),
line_id     NUMBER(10) SORT,
delivery_dt DATE SORT)
TABLESPACE uwdata
HASHKEYS 9
SIZE 750
HASH IS program_id;

CREATE TABLE shc_airplane (
program_id  NUMBER(3),
line_id     NUMBER(10) SORT,
delivery_dt DATE SORT,
customer_id VARCHAR2(3),
order_dt    DATE)
CLUSTER sorted_hc (program_id, line_id, delivery_dt);
```

Demo

Chained Rows

- A chained row is a row that is written to two or more blocks
- Row chaining inevitably leads to more I/O than would be required if the entire row fit within a single block

```
SQL> @?/rdbms/admin/utlchn1.sql

SQL> ANALYZE TABLE t LIST CHAINED ROWS INTO chained_rows;

SQL> SELECT sys_op_rpb(rowid), table_name, head_rowid, analyze_timestamp
  2  FROM chained_rows;

SQL> SELECT rowid, dbms_rowid.rowid_block_number(rowid) BN, sys_op_rpb(rowid), length(col1), length(col2)
  2  FROM t;
```

Index Use and Abuse

- What is the right number of indexes on a table?
- It depends but a number between 0 and 6 is reasonable
- One way to determine if an index is being used is to monitor usage ... but it can be misleading because while the index may not be read ... the stats on it may be of great value to the optimizer in choosing the correct execution path
  - If an index that is not used is dropped it may affect execution plans
  - If an index that is not used is altered it may affect execution plans
  - If an index that is not used is created it may affect execution plans

```
SQL> ALTER INDEX ix_index_demo_gender_state MONITORING USAGE;

SQL> SELECT *
  2  FROM v$object_usage;
```

Sequence Caching

- If you see something like this invest some time in sequence tuning
- Cache Size specifies how many values of the sequence the database preallocates and keeps in memory for faster access
- In the event of a system failure all unused cached sequence values are lost
- There is substantial overhead, and locking each time the sequence pre-allocates a new set of values
- Oracle's default cache value of 20 is a bad joke and should never be used for anything other than demos of what is wrong with a value of 20

```
SQL> SELECT order_flag, cache_size, COUNT(*)
  2  FROM dba_sequences
  3  WHERE sequence_owner NOT LIKE '%SYS%'
  4  GROUP BY order_flag, cache_size
  5* ORDER BY order_flag, cache_size;

O CACHE_SIZE   COUNT(*)
- ---------- ----------
N          0          7
N          9          4
N         10          4
N         20         52
Y          0         52
Y         20          8
Y        100          1
```

Sequence Ordering

- By default sequences are guaranteed to create unique numbers but when you force ordering you force serialization which will always negatively impact performance
- Do not order sequences unless the application demands it

Sequences and RAC

- On RAC clusters watch for the DFS Lock Handle wait as an indication of incorrectly sized caching: If found increase the caching size
- If a sequence is used to create a surrogate key and there are 1,000 inserts each second caching 20 values means that 500 times a second the cache will need to be refreshed
- This will negatively impact performance
- For this example cache 5,000+ values

Binary XML Partitioning Example

- High performance means not running version 7.3.4 code in 11.2.0.4+

```
CREATE TABLE orders OF XMLType
 XMLTYPE STORE AS BINARY XML
 VIRTUAL COLUMNS (
  SITE_ID AS (XMLCast(XMLQuery('/Order/@SiteId' PASSING OBJECT_VALUE RETURNING CONTENT) AS NUMBER)))
 PARTITION BY RANGE (site_id) (
 PARTITION p1 VALUES LESS THAN (10),
 PARTITION p2 VALUES LESS THAN (20),
 PARTITION pm VALUES LESS THAN (MAXVALUE));

 DECLARE
  x XMLTYPE;
 BEGIN
   x := XMLTYPE('<?xml version="1.0" encoding="utf-8"?>
          <Order orderId="1" orderRevision="1" orderTimeStamp="01-JAN-2012">
            <OrderHeader>
              <AlternateIds>
                <AlternateId altIdType="SiteId">12</AlternateId>
                <AlternateId altIdType="MerchantOrderNumber">Merch</AlternateId>
                <AlternateId altIdType="MarketplaceOrderNumber">Place</AlternateId>
                <AlternateId altIdType="CustomerReferenceId">Ref</AlternateId>
                <AlternateId altIdType="CartId">Cart</AlternateId>
                <AlternateId altIdType="SessionId">1</AlternateId>
              </AlternateIds>
            </OrderHeader>
          </Order>');
   INSERT INTO orders VALUES (x);
 END;
 /
...
```

More ...

Jonathan Lewis' Rules for Hints

1. **Don't**

2. **If you must use hints, then assume you've used them incorrectly**

3. **On every patch or upgrade to Oracle, assume every piece of hinted SQL is going to do the wrong thing**

   Because of (2) above; you've been lucky so far, but the patch/upgrade lets you discover your mistake

4. **Every time you apply some DDL to an object that appears in a piece of hinted SQL assume that the hinted SQL is going to do the wrong thing**

   Because of (2) above; you've been lucky so far, but the structural change lets you discover your mistake

Aliasing and Fully Qualified Names

- When you do not use fully qualified names Oracle must do the work for you
- You write code once ... the database executes it many times

```
SELECT DISTINCT s.srvr_id
FROM servers s, serv_inst i
WHERE s.srvr_id = i.srvr_id;

SELECT DISTINCT s.srvr_id
FROM uwclass.servers s, uwclass.serv_inst i
WHERE s.srvr_id = i.srvr_id;
```

Advanced Rewrite

- The DBMS_ADVANCED_REWRITE package is fully documented and supported
- With Advanced Rewrite you can replace "bad" SQL with "good" SQL without having access to, or fixing, application code

```
ALTER SYSTEM SET query_rewrite_integrity = 'TRUSTED'
COMMENT='Permanent Change Rewrite From ENFORCED to TRUSTED'
SCOPE=BOTH;



dbms_advanced_rewrite.declare_rewrite_equivalence(
name              VARCHAR2,
source_stmt       CLOB,
destination_stmt  CLOB,
validate          BOOLEAN  := TRUE,
mode              VARCHAR2 := 'TEXT_MATCH');
```

- In 12c look at DBMS_SQL_TRANSLATOR

Automatic Memory Management

- In Oracle Database 10g memory management was ASMM (Automatic Shared Memory Management)
- In Oracle 11g through 11.2.0.3 the default was AMM (Automatic Memory Management) which was found to produce too many memory resizings and, as a result, performance issues
- As of Database 11.2.0.4 Oracle has reverted to ASMM as the preferred algorithm
- To determine the number of resize operations run the following SQL statement and if using AMM convert back to ASMM: If using ASMM optimize your memory parameters to reduce their frequency

```
SELECT TRUNC(start_time), status, oper_type, oper_mode, parameter
FROM v$sga_resize_ops
WHERE initial_size <> final_size;
```

Block Change Tracking (1:3)

- If a database is small it is possible to perform a full RMAN backup every night: This is rarely possible or necessary
- When configuring incremental backups we generally configure a separate back every 15 to 120 minutes for archive logs and perform a nightly incremental backup
  - Level 0 once or twice a week
  - Level 1 every night on which a Level 0 is not taken
- Incremental Level 1 backups will always benefit from block change tracking
- To enable block change tracking you must create a block change tracking file

```
conn / as sysdba

SQL> SELECT filename, status, bytes
  2  FROM v$block_change_tracking;


FILENAME                                           STATUS          BYTES
-------------------------------------------------- ---------- ----------
                                                   DISABLED
```

Block Change Tracking (2:3)

- The size of the file created is determined by the number of blocks in your database's data files: The more blocks the larger the file

```
SQL> ALTER DATABASE ENABLE BLOCK CHANGE TRACKING
  2* USING FILE 'c:\app\oracle\fast_recovery_area\ORABASE\bctf01.log';
Database altered.

SQL> SELECT filename, status, bytes
  2  FROM v$block_change_tracking;

FILENAME                                                 STATUS          BYTES
-------------------------------------------------------- ---------- ----------
C:\APP\ORACLE\FAST_RECOVERY_AREA\ORABASE\BCTF01.LOG       ENABLED      11599872
```

Block Change Tracking (3:3)

- When Block Change Tracking is enabled the instance spawns the CTWR process which is responsible for writing log file entries

```
SQL> SELECT *
  2  FROM v$sgastat
  3  WHERE name LIKE '%CTWR%';

   INST_ID POOL         NAME                            BYTES     CON_ID
---------- ------------ ------------------------------ ---------- ----------
         1 large pool   CTWR dba buffer                1728512          1

SQL> SELECT inst_id, sid, program, status
  2  FROM v$session
  3  WHERE program LIKE '%CTWR%';

   INST_ID        SID PROGRAM                                                  STATUS
---------- ---------- ------------------------------------------------------- --------
         1        248 ORACLE.EXE (CTWR)                                       ACTIVE
```

Block Corruption (2:2)

- If corruption is found use the following SQL to identify the corrupted segment(s)

```
SELECT de.owner, de.segment_name, de.segment_type
FROM dba_extents de, v$database_block_corruption vdbc
WHERE de.file_id = vdbc.file#
AND vdbc.block# BETWEEN de.block_id AND (de.block_id+(de.blocks-1));
```

- To eliminate the corruption
  - DBMS_REPAIR
  - Drop and rebuild the segment
  - Open an SR with Oracle

Connection Pooling and Middle Tier Caching

- Opening and maintaining a database connection for each user
  - Especially when dynamically creating sessions for requests made to a dynamic database-driven web application are costly and waste resources
- A cache with multiple persistent database connections
- After a connection is created, it is placed in the pool, and it is reused so that a new connection does not have to be established for each new session or inquiry
- Each persistent connection can be reused when a future request to the database is made
- Used to enhance the performance of executing commands in a database

Implicit Casts

- Code that does not correctly define data types will either fail to run or run very inefficiently

  The following example shows both the correct way and the incorrect way to work with dates. The correct way is to perform an explicit cast

```
SQL> create table t (
  2  datecol date);

Table created.

SQL> insert into t values ('01-JAN-2015');

1 row created.

SQL> insert into t values (TO_DATE('01-JAN-2015'));

1 row created.
```

In Oracle dates are dates ... not strings. Similarly numbers should either be explicitly cast with TO_NUMBER or only processed using numeric variables.

Implicit Commits

- An explicit commit must be present at the end of any transaction
- You can not rely on an ODBC or JDBC driver configuration to commit your changes in a reliable and consistent way
- Similarly ... you must explicitly rollback in exception handlers as the ODBC driver may perform an unintended commit

```
CREATE OR REPLACE PROCEDURE test AUTHID DEFINER IS
BEGIN
  FOR i IN 1..10 LOOP
    INSERT INTO t
    (testcolumn)
    VALUES
    (i);
  END LOOP;

  COMMIT;
EXCEPTION
  WHEN dup_val_on_index THEN
    ROLLBACK;
END;
/
```

Inactive Sessions

- Ignoring, for the moment, issues of grammar and spelling ... clearly there is a technology problem

Attachments

Hi Bob

we have killed 250 sessions only that are blocking other sessions.

1. What session was causing the blocking lock? The client would normally like to know what SCHEMA and WHAT WAS THE LAST SQL STATEMENT ISSUED...
Ans:blocking session

```
id's: (25,153,889,967,1098,1356,552,554,620,740,949,1172,2,224,410,596,610,635,814,817,937,1089,1177,
1195,1307,1316,1458,24,53,68,304,467,509,556,825,970,1035,1119,1925,38,686,830,832,1414,1511,333,506,
792,932,942,115,1008,1014,19,313,1198,1202,1224,1376);
```

Please Note: Not killed any active sessions for these blocking sessions. Killed only INACTIVE sessions.

Invalid Objects and Unusable Indexes

- Look for newly invalidated objects and unusable indexes

```
conn / as sysdba

SELECT owner, object_type, COUNT(*)
FROM dba_objects_ae
WHERE status = 'INVALID'
GROUP BY owner, object_type;

SELECT owner, table_name, index_name
FROM dba_indexes
WHERE status = 'UNUSABLE';
```

Java Fetch Size

- Most applications that connect to the Oracle database do so through a driver such as JDBC or ODBC
- By default these drivers are configured with an array size, usually about 10 that lead to slow response due to their inefficiency
- An optimal array size is likely between 100 and 250 yielding 1/10th to 1/25th the number of I/O requests
- In JDBC the array size is set with the **`java.sql.Statement.setFetchSize()`** parameter
- The following are other parameters that can be used to optimize application performance
  - User **`statement.cancel or Statement.setQueryTimeout`**
  - **`jdbc:oracle:thin:@(DESCRIPTION ...(SDU=32767)...)`**

"Jobs" and Human Nature

- Jobs that do not real work can be very expensive
  - Find them
  - Try to identify an owner
  - Try to fix them
  - If you can't ... disable them

```
SQL> SELECT owner, job_name, job_type, trunc(start_date) SDATE, trunc(next_run_date) nxtrun, failure_count
  2  FROM dba_scheduler_jobs
  3* WHERE failure_count <> 0;

OWNER           JOB_NAME                       STATE      SDATE                NXTRUN               FAILURE_COUNT
--------------- ------------------------------ ---------- -------------------- -------------------- -------------
SYS             SM$CLEAN_AUTO_SPLIT_MERGE      SCHEDULED  14-MAR-2011 00:00:00 14-AUG-2013 00:00:00            17
SYS             RSE$CLEAN_RECOVERABLE_SCRIPT   SCHEDULED  14-MAR-2011 00:00:00 14-AUG-2013 00:00:00            20
SYS             DRA_REEVALUATE_OPEN_FAILURES   SCHEDULED                                                       10
ORACLE_OCM      MGMT_CONFIG_JOB                SCHEDULED                                                        4
EXFSYS          RLM$SCHDNEGACTION              SCHEDULED  13-AUG-2013 00:00:00 13-AUG-2013 00:00:00             3
EXFSYS          RLM$EVTCLEANUP                 SCHEDULED  27-APR-2011 00:00:00 13-AUG-2013 00:00:00             2
RDBA5           LONG_RUN_SESS_JOB              SCHEDULED  12-AUG-2013 00:00:00 13-AUG-2013 00:00:00             1
EISAI_PROD_TMS  POPULATE_MORGAN_CATALOG        DISABLED   01-JUN-2009 00:00:00 08-AUG-2013 00:00:00          2559
```

Redo Log Switch Anomalies (1:3)

- Redo log switches should occur between 4 and 12 times per hour (once every 5 to 15 minutes) to balance performance with transaction security
- Monitoring redo log switches aggregated by the hour can alert you to incorrectly sized redo logs, behavioral changes, unexpected workloads, and outages
- The SQL statement on the following page can be used to monitor the switches on stand-alone and RAC One-Node systems and provide general guide with RAC clusters

Redo Log Switch Anomalies (2:3)

```
SELECT TO_CHAR(first_time,'MMDD') MMDD,
TO_CHAR(SUM(DECODE(TO_CHAR(first_time,'HH24'),'00',1,0)),'99') "00",
TO_CHAR(SUM(DECODE(TO_CHAR(first_time,'HH24'),'01',1,0)),'99') "01",
TO_CHAR(SUM(DECODE(TO_CHAR(first_time,'HH24'),'02',1,0)),'99') "02",
TO_CHAR(SUM(DECODE(TO_CHAR(first_time,'HH24'),'03',1,0)),'99') "03",
TO_CHAR(SUM(DECODE(TO_CHAR(first_time,'HH24'),'04',1,0)),'99') "04",
TO_CHAR(SUM(DECODE(TO_CHAR(first_time,'HH24'),'05',1,0)),'99') "05",
TO_CHAR(SUM(DECODE(TO_CHAR(first_time,'HH24'),'06',1,0)),'99') "06",
TO_CHAR(SUM(DECODE(TO_CHAR(first_time,'HH24'),'07',1,0)),'99') "07",
TO_CHAR(SUM(DECODE(TO_CHAR(first_time,'HH24'),'08',1,0)),'99') "08",
TO_CHAR(SUM(DECODE(TO_CHAR(first_time,'HH24'),'09',1,0)),'99') "09",
TO_CHAR(SUM(DECODE(TO_CHAR(first_time,'HH24'),'10',1,0)),'99') "10",
TO_CHAR(SUM(DECODE(TO_CHAR(first_time,'HH24'),'11',1,0)),'99') "11",
TO_CHAR(SUM(DECODE(TO_CHAR(first_time,'HH24'),'12',1,0)),'99') "12",
TO_CHAR(SUM(DECODE(TO_CHAR(first_time,'HH24'),'13',1,0)),'99') "13",
TO_CHAR(SUM(DECODE(TO_CHAR(first_time,'HH24'),'14',1,0)),'99') "14",
TO_CHAR(SUM(DECODE(TO_CHAR(first_time,'HH24'),'15',1,0)),'99') "15",
TO_CHAR(SUM(DECODE(TO_CHAR(first_time,'HH24'),'16',1,0)),'99') "16",
TO_CHAR(SUM(DECODE(TO_CHAR(first_time,'HH24'),'17',1,0)),'99') "17",
TO_CHAR(SUM(DECODE(TO_CHAR(first_time,'HH24'),'18',1,0)),'99') "18",
TO_CHAR(SUM(DECODE(TO_CHAR(first_time,'HH24'),'19',1,0)),'99') "19",
TO_CHAR(SUM(DECODE(TO_CHAR(first_time,'HH24'),'20',1,0)),'99') "20",
TO_CHAR(SUM(DECODE(TO_CHAR(first_time,'HH24'),'21',1,0)),'99') "21",
TO_CHAR(SUM(DECODE(TO_CHAR(first_time,'HH24'),'22',1,0)),'99') "22",
TO_CHAR(SUM(DECODE(TO_CHAR(first_time,'HH24'),'23',1,0)),'99') "23"
FROM v$log_history
GROUP BY TO_CHAR(first_time,'MMDD')
ORDER BY 1;
```

Redo Log Switch Anomalies (3:3)

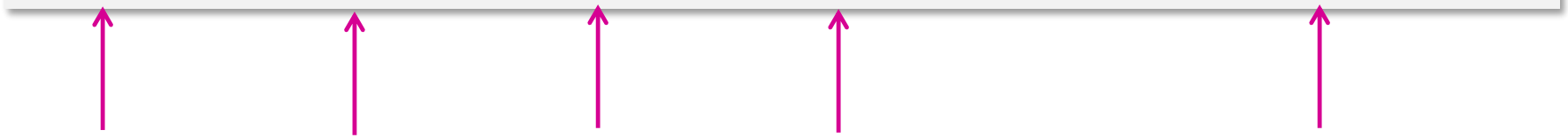
MMDD	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0609	16	11	9	8	8	10	12	8	8	10	8	10	14	10	11	15	15	8	12	8	7	6	9	7
0610	13	12	8	9	7	6	11	9	6	8	7	8	12	6	7	6	8	7	10	7	4	4	4	5
0611	12	8	5	9	9	7	11	7	6	7	8	5	12	9	10	8	9	12	12	10	6	6	9	8
0612	13	12	7	9	7	9	10	10	7	7	9	8	11	7	7	8	7	7	11	9	5	6	8	7
0613	12	11	7	8	8	7	13	7	9	7	8	7	13	10	9	8	8	8	11	8	7	5	7	6
0614	15	10	9	9	8	9	13	9	9	7	11	13	11	9	8	9	13	9	12	9	7	9	7	7
0615	15	10	10	8	10	9	12	8	9	8	9	7	13	6	8	7	7	7	15	10	7	7	7	5
0616	13	8	8	7	7	6	10	8	11	7	8	6	11	7	12	13	13	14	13	9	9	9	7	8
0617	15	13	10	9	8	9	16	8	8	10	9	10	16	11	10	10	8	11	13	8	9	9	7	9
0618	12	13	15	15	13	13	15	13	9	12	8	11	14	9	10	9	9	8	14	9	8	8	9	8
0619	16	11	10	11	9	9	13	12	10	9	12	12	17	8	9	9	11	11	14	9	9	11	10	12
0620	19	15	11	10	10	10	19	11	9	9	9	9	13	7	15	10	11	11	12	10	9	11	11	10
0621	13	16	11	9	10	13	16	8	14	9	11	12	17	10	10	11	8	11	14	8	11	14	8	11
0622	16	13	13	11	11	9	16	9	9	11	10	11	17	10	9	10	10	10	13	14	9	10	10	8
0623	19	13	12	13	13	11	16	12	11	11	11	11	16	9	10	13	2	14	14	8	9	8	8	8
0624	14	9	9	9	7	9	11	8	8	7	8	8	14	7	8	7	9	3	6	0	0	0	0	0
0625	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0
0626	0	1	0	0	0	0	0	0	0	4	0	0	0	2	2	3	2	7	5	6	1	0	0	0
0627	3	10	0	0	0	5	0	1	10	0	0	0	0	0	1	0	1	0	2	5	3	7	1	0
0629	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	4	0	0	6	7	6
0630	7	4	23	19	9	10	5	6	7	17	19	17	15	17	15	43	40	32	17	15	14	20	13	15
0701	15	12	14	12	13	12	13	17	15	17	20	20	18	18	17	15	14	13	10	10	15	15	13	19
0702	21	22	20	18	14	14	12	13	11	11	14	14	14	10	9	10	9	10	11	9	11	9	10	12
0703	9	13	10	17	14	17	15	17	23	20	19	20	17	19	16	17	15	17	15	15	15	16	16	18
0704	22	19	19	18	16	15	13	13	14	11	13	10	12	14	10	12	14	11	9	11	12	13	12	9
0705	14	13	9	11	10	12	13	11	11	8	10	10	11	11	11	12	10	10	9	10	8	9	12	7
0706	14	15	11	12	9	15	13	12	12	9	12	14	12	12	12	12	13	11	8	9	12	13	2	0
0707	0	0	1	0	3	15	10	10	7	8	10	11	12	8	6	9	13	12	9	8	9	8	10	10
0708	16	9	8	15	10	11	9	8	8	14	9	10	10	8	8	14	15	10	9	9	8	9	10	10
0709	13	12	9	10	10	9	9	10	11	11	8	9	9	8	9	13	8	9	6	9	9	11	10	9
0710	12	10	9	10	9	12	9	8	8	11	7	10	11	9	9	13	10	9	8	9	11	12	10	10
0711	15	12	9	13	9	12	8	10	11	13	9	8	10	9	8	12	11	12	9	9	10	11	10	8
0712	13	12	10	13	10	10	9	7	10	11	9	10	12	12	12	15	12	9	8	9	11	12	12	12
0713	14	12	12	11	10	10	12	12	12	15	10	11	11	10	4	5	15	14	10	9	8	8	13	6
0714	12	12	9	9	11	10	10	9	10	9	14	7	7	8	8	9	14	9	9	10	12	8	13	10
0715	10	10	9	14	12	15	12	14	13	15	10	11	9	4	8	6	8	7	6	7	8	8	8	8
0716	10	11	9	8	8	9	9	6	6	7	7	12	7	9	15	14	13	16	12	14	11	9	6	7
0717	10	10	9	9	9	10	12	14	11	10	12	9	8	12	7	3	0	0	0	0	0	0	0	0

Redo Log Switch Anomalies (4:4)

- Another real-world example of redo log switches helping to focus attention

RAC Server Node 1

MMDD	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0804	0	0	0	0	0	0	0	0	0	0	0	0	5	32	18	65	91	13	12	20	84	9	14	9
0805	137	112	26	27	141	17	21	9	85	13	21	17	96	23	23	24	91	13	11	21	86	11	14	9
0806	151	111	21	24	96	41	50	14	84	22	20	22	91	18	17	18	92	24	10	11	83	9	14	20
0807	139	100	32	30	99	43	49	19	105	17	31	14	76	23	27	25	111	20	15	18	86	13	13	10
0808	145	99	29	30	109	52	48	11	102	25	47	24	101	23	20	23	117	31	30	16	91	12	11	9
0809	123	83	65	37	93	17	25	10	102	23	44	25	111	37	24	29	98	19	29	16	92	16	15	9
0810	169	120	52	32	125	58	38	9	109	17	26	14	104	13	17	15	93	13	16	11	61	10	10	9
0811	107	82	51	34	85	17	22	10	73	10	12	11	92	32	13	69	65	11	11	10	60	9	12	9
0812	149	121	26	15	70	16	24	11	95	34	15	18	34	67	21	21	87	11	13	9	77	9	14	9
0813	115	76	55	56	27	9	9	9	11	9	9	0	0	0	0	0	0	0	0	0	0	0	0	0



60 corresponds to one change per minute ... the ideal range is 4 to 12
Addressed by resizing redo logs from 400MB to 4GB
And rescheduling many of the jobs

Scheduler Job Failures

- It is a common occurrence that DBMS_SCHEDULER jobs fail and no one notices because
  - The job has been automatically running for a long time but no one using the output
  - Job failures are not reported in the alert log
- The following example shows how to identify failed jobs
- The job shown was disabled on discovering the failure count

```
SQL> SELECT owner, job_name, job_type, trunc(start_date) SDATE, trunc(next_run_date) nxtrun, failure_count
  2  FROM dba_scheduler_jobs
  3* WHERE failure_count <> 0;

OWNER           JOB_NAME                        STATE      SDATE                NXTRUN               FAILURE_COUNT
--------------- ------------------------------- ---------- -------------------- -------------------- -------------
SYS             SM$CLEAN_AUTO_SPLIT_MERGE       SCHEDULED  14-MAR-2011 00:00:00 14-AUG-2013 00:00:00            17
SYS             RSE$CLEAN_RECOVERABLE_SCRIPT    SCHEDULED  14-MAR-2011 00:00:00 14-AUG-2013 00:00:00            20
SYS             DRA_REEVALUATE_OPEN_FAILURES    SCHEDULED                                                        10
ORACLE_OCM      MGMT_CONFIG_JOB                 SCHEDULED                                                         4
EXFSYS          RLM$SCHDNEGACTION               SCHEDULED  13-AUG-2013 00:00:00 13-AUG-2013 00:00:00             3
EXFSYS          RLM$EVTCLEANUP                  SCHEDULED  27-APR-2011 00:00:00 13-AUG-2013 00:00:00             2
RDBA5           LONG_RUN_SESS_JOB               SCHEDULED  12-AUG-2013 00:00:00 13-AUG-2013 00:00:00             1
EISAI_PROD_TMS  POPULATE_MORGAN_CATALOG         DISABLED   01-JUN-2009 00:00:00 08-AUG-2013 00:00:00          2559
```

Objects Named With Keywords

- How to test a word to determine whether it can be used

```
SQL> SELECT keyword
   2 FROM v$reserved_words
   3 WHERE keyword LIKE 'ID%';

KEYWORD
------------------------------
ID
IDENTIFIED
IDGENERATORS
IDLE_TIME
IDENTITY
IDENTIFIER

6 rows selected.
```

Columns should never be named "ID." Best practice is to identify the nature of the ID, for example, MEMBER_ID or ORDER_ID. This holds true for columns such as COMMENTS and DATES.

Where Are Sorts Taking Place?

- In memory or on disk?

```
SELECT a.value "Disk Sorts", b.value "Memory Sorts",
ROUND((100*b.value)/DECODE((a.value+b.value), 0,1,(a.value+b.value)),2) "Pct Memory Sorts"
FROM v$sysstat a, v$sysstat b
WHERE a.name = 'sorts (disk)'
AND b.name = 'sorts (memory)';
```

Undo Tablespace Size and Retention

- Check undo tablespace size and resize in accordance with any advisory

```
set serveroutput on

DECLARE
 prob VARCHAR2(100);
 reco VARCHAR2(100);
 rtnl VARCHAR2(100);
 retn PLS_INTEGER;
 utbs PLS_INTEGER;
 retv PLS_INTEGER;
BEGIN
  retv := dbms_undo_adv.undo_health(prob, reco, rtnl, retn, utbs);
  dbms_output.put_line('Problem: ' || prob);
  dbms_output.put_line('Recmmnd: ' || reco);
  dbms_output.put_line('Rationl: ' || rtnl);
  dbms_output.put_line('Retentn: ' || TO_CHAR(retn));
  dbms_output.put_line('UTBSize: ' || TO_CHAR(utbs));
END;
/
```

PL/SQL

Debug Mode Compilation

- It is "Best Practice" to always build new schemas, and compile database code, using SQL*Plus running pure ASCII scripts
- Many of the GUI tools used by Developers, and sometimes by DBAs, by default compile PL/SQL objects in debug mode
- If debug mode objects are found they should be recompiled using SQL*Plus

```
SELECT owner, name, type
FROM dba_plsql_object_settings
WHERE plsql_debug='TRUE'
ORDER BY 1,3,2;
```

PL/SQL Warnings

- New warnings in 10g and 11g: No new warnings in 12c
- Invaluable for finding suboptimal code, use of reserved words, inappropriate usages, and orphans Embedded SQL vs. Database APIs
  - Severe
    - 5018 - omitted optional AUTHID clause
    - 5019 - deprecated language element
    - 5020 - parameter name must be identified
  - Informative
    - 6016 - native code generation turned off (size/time)
    - 6017 - operation will raise an exception
    - 6018 - an infinity or NaN value computed or used
  - Performance
    - None

Wrap-Up

Summary

- If your problem is not caused by slow hardware then faster hardware will just create a problem faster
- Gather information from a date-time range when everything was good and compare with the date-time range when the problem was observed
- If the problem is slow SQL ... is the SQL statement itself slow or is it the entire environment that should be examined?
- If the problem is related to CPU ... do you need more/faster CPUs or a better design?
- If the problem is related to I/O ... do you need a new SAN, an improved layout, or better indexing?
- If the problem is related to network bandwidth ... do you need new NIC cards, jumbo frames, or to stop network virtualization? Yes you probably do

Conclusion

- Do not tune by guessing
- Do not blame everything on bad SQL statements
- Until you can prove that the root cause is the SQL
- Use the scientific method
  - Construct a hypothesis
  - Test the hypothesis by doing experiments
  - Validate your conclusion
- Only rewrite SQL after you have established conclusively that the root cause is the way a statement has been written and that rewriting the SQL statement will address ALL issues

- You will most likely not be rewriting a lot of SQL statements

Conclusion

- Do not tune by guessing
- Do not blame everything on bad SQL statements
- Until you can prove that the root cause is the SQL
- Use the scientific method
  - Construct a hypothesis
  - Test the hypothesis by doing experiments
  - Validate your conclusion
- Only rewrite SQL after you have established conclusively that the root cause is the way a statement has been written and that rewriting the SQL statement will address ALL issues

```
SELECT more_information
FROM experience
WHERE topic = 'Performance Tuning'
AND database = 'Innocent Bystander';
```

- You will most likely not be rewriting a lot of SQL statements

```
*
ERROR at line 1:
ORA-00028: your session has been killed
```

Thank You

Daniel A. Morgan
email: dmorgan@forsythe.com
mobile: +1 206-669-2949
skype: damorgan11g
twitter: @meta7solutions