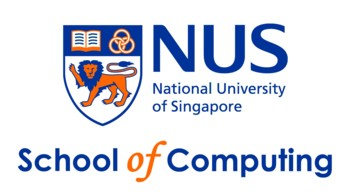
CS5226 – DATABASE TUNING

# **Semester 2 - AY 2012/013**



|  |  |
| --- | --- |
| Name of Student | Matriculation Number |
| Aasim Shaikh | A0106467A |
| Aishwarya | A0096133H |
| Rajesh Maddala | A0095940X |
| Shailee Dipak Mehta | A0096013M |

April 12, 2013

Database Performance Dashboard



## WORKLOAD DISTRIBUTION

|  |  |  |
| --- | --- | --- |
| TEAM MEMBER NAME | MATRICULATION NO | PERCENTAGE |
| Aasim Shaikh | A0106347A | 25% |
| Aishwarya | A0096133H | 25% |
| Rajesh Maddala | A0095940X | 25% |
| Shailee Dipak Mehta | A0090013M | 25% |

# INTRODUCTION

Database Tuning is the activity of making a database application run efficiently and optimally, i.e. to obtain higher throughput, albeit at the cost of lower response time for time-critical applications. In this project a **Database Performance Dashboard** has been developed that aids the database administrators to monitor the health of the database and support performance investigations. The performance dashboard provides a one-stop overview of various key performance indicators (KPIs) on database performance. Users can then make use of the high-level information to further zoom in to identify the performance issues, and tune the system to improve performance.

# PROJECT OVERVIEW

The database performance dashboard is a web application that has a 3-tiered architecture which consists of a presentation/web layer, business layer and a data layer. The application is modeled around an **Inventory Management System** wherein the relations that contain information about customers, materials, suppliers, order and invoice are stored in an Oracle database. The webApp connects with this database using the JDBC framework which facilitates the storage and retrieval of Java domain objects via [Relational Mapping](http://www.hibernate.org/hibernate/about/orm.html). Native queries are run via a JDBC connection between the application and the database and query outputs are mapped to the members of a Java Data Access Object (DAO). Java Server Pages (JSP) has been used to develop the webApp.

The application performs a series of periodic updates on the content of the database while accumulating the database statistics. These statistics are then compiled and rendered in the UI so that database administrators can stay informed about the performance of the database and take necessary action when there is a deviation from normal performance, if any.

# SYSTEM DESIGN AND ARCHITECTURE

WEB APPLICATION DESIGN: The web application interface makes use of JSP as server side scripting language to interface with the Apache Server and Oracle database. Jquery/Javascript has been used for client side scripting in order to handle real time asynchronous updates. Jquery also takes care of the navigation of HTML documents, adds Ajax interactions to the web pages and performs animation and events handling.

DATABASE DESIGN:Our database consists of a set of relations which contain data about theatres, movies, screens, orders placed by users as well as the dashboard performance parameters. The dashboard table is updated continuously in order to maintain an up-to-date snapshot of database performance parameters.

The E-R diagram of the database schema is provided as under:



# PERFORMANCE PARAMETERS

This section provides a description of the various Key Performance Parameters that have been analyzed as a part of our application.

## BUFFER CACHE

Oracle data is stored in Buffer cache within the SGA for each instance to minimize physical disk I/O by holding (buffering) copies of requested data blocks in memory. Thus, all the data returned to the users is from this Buffer cache. Data stored in this cache is catered to users at very high speed i.e. memory access speed as compared to a physical disk I/O. A large performance penalty caused due to physical disk I/O makes the tuning of Buffer cache very crucial for the DBA.

Whenever a user queries some data, Oracle first checks for that data in buffer cache. If the data is stored in the cache it is returned to the user instantly. When the data is buffered in the cache, it leads to reading the data from physical disk into an available buffer in the cache.

### Buffer Cache Hit Rate

Buffer Cache Hit Rate indicates the overall percentage of data requests that are catered directly from the Buffer Cache. This can be calculated using the following formula:

Cache Hit Ratio = 100 \* (1 - physical reads/logical reads)

Note: The physical reads here indicates the Buffer Cache Misses and logical reads indicates the total data requests.

If the buffer cache is very small then, the cache hit rate will be very small indicating that more physical disk I/O would be done. If the buffer cache is too big, then proportions of buffer cache will be under-utilized and memory resources are wasted.

A properly sized buffer cache can usually yield a cache hit ratio over 90%, meaning that nine requests out of ten are satisfied without going to disk.

Oracle maintains statistics of buffer cache hits and misses. The following query will show you the overall buffer cache hit ratio for the entire instance since it was started:

***SELECT***

***(a.value + b.value - c.value) / (a.value + b.value)***

***FROM v$sysstat a, v$sysstat b, v$sysstat c***

***WHERE a.name = 'db block gets'***

***AND b.name = 'consistent gets'***

***AND c.name = 'physical reads'***

Tuning the buffer cache for optimum performance usually involves adding buffers to the cache until the hit ratio has been maximized. The DB\_BLOCK\_BUFFERS parameter in the parameter file determines the size of the buffer cache for the instance. You can change the size of the buffer cache by editing the DB\_BLOCK\_BUFFERS parameter in the parameter file and restarting the instance.

## SHARED POOL

At the time of Oracle startup, a RAM area from the RAM heap is created which is known as shared pool. This area is the most essential component of System Global Area (SGA).This is because it is the *shared\_pool\_size*parameter which controls various sub-areas within the SGA. All these sub-areas are used for different purposes. Hence to ensure health of database, Database Administrator needs to monitor the Shared pool contention.

Whenever Oracle tries to execute a query, it ensures that the fired SQL statement is not parsed every time it is executed. Shared pool RAM acts as a buffer to store all these SQL statements.

The main areas in shared pool RAM within the SGA are as follows:

* **Library Cache** is one of the sub-areas in the shared pool RAM within the SGA. Library Cache contains the currently fired SQL statement execution plan information. It collects interprets and execute all the SQL statements fired on the Oracle database.
* **Data Dictionary cache** also known as **row cache** is the area where all the referential integrity, indexing information, table definitions and other metadata about Oracle’s internal system tables. It stores all the data dictionary related information in shared pool RAM for easy and quick access.
* Shared Pool area also stores information about the current session e.g. v$session view of Oracle\*’Net users.

If the size of the shared pool RAM is low, it may cause gradual increase in library cache reloads, row cache reloads and shared pool latch contention leading to aggravate the health and performance of the database. It produces the error: "ORA-04031: Out of shared pool memory".

### Managing the SHARED\_POOL\_SIZE parameter

#### Shared Pool Reloads Ratio/Library Cache Miss Ratio

When an Oracle SQL statement is fired, it is first compiles in two phases. The first phase parses the SQL statement and the second phase executes it. Oracle searches the Library cache while parsing the SQL statement. If it does not find any record for the current SQL statement, it will allocate an area in the shared pool RAM within the library cache and then parse the SQL statement. Thus, when executing this statement, Oracle checks whether this SQL statement already exists in the library cache. If not, then it will reparse the statement as mentioned above and execute it.

We can find out the shared pool reload ratio or library cache miss ratio(both are similar) which is the ratio of library cache reloads to the sum of pins using the following query:

***SELECT***

***SUM(PINS) "EXECUTIONS",***

***SUM(RELOADS) "CACHE MISSES WHILE EXECUTING",***

***round(((sum(RELOADS)/sum(PINS))\*100),2) "SHARED POOL RELOAD RATIO"***

***FROM***

***V$LIBRARYCACHE;***

This ratio of cache misses to execution indicates the number of times Oracle fails to search the library cache for the current SQL/PLSQL statement and reparse the statement in to the library cache instead of being reused. Thus if this value is greater than 1, then the value of **SHARED\_POOL\_SIZE parameter must be increased in the init.ora file.**

#### Library Cache Hit Rate

The library cache hit ratio indicates how many times Oracle hits the library cache to parse a particular SQL/PLSQL statement without a cache miss occurring. Following query gives the library cache hit rate:

***SELECT***

***round((1-((sum(RELOADS)/sum(PINS))))\*100,2) "LIBRARY CACHE HIT RATE"***

***FROM***

***V$LIBRARYCACHE;***

Thus, this rate should be 95% to ensure the health and performance of the database. This means that the data stored in the library cache were never aged or invalidated. Thus if this rate decreases, then the value of **SHARED\_POOL\_SIZE parameter must be adjusted in the init.ora file.**

#### Data Dictionary Cache Hit Rate

Data Dictionary cache contains all the referential integrity, indexing information, table definitions and other metadata about Oracle’s internal system tables. Hence when the Oracle is started, the initial value in data dictionary cache is less but as more and more data is queried, this cache is filled. Data dictionary cache hit rate indicates the number of times Oracle hits the dictionary cahce to fetch data without a cache miss occurring. Following query returns the Dictionary Cache Hit Rate from v$ROWCACHE system table:

***SELECT***

***round((1-(sum(getmisses)/(sum(gets) + sum(getmisses))))\*100,2) “DICTIONARY\_HIT\_RATE”***

***FROM***

***v$ROWCACHE;***

To ensure healthiness of the database, one should not allow this rate to fall below 95%. If it goes below that then,then the value of **SHARED\_POOL\_SIZEparameter must be increased in the init.ora file.**

#### Shared Pool Free Ratio

Shared Pool Free Ratio indicates the ratio of the shared pool RAM area which is not currently in use. If a large portion of the shared pool are is always free then a suggestion is to reduce the size of the shared pool. A low shared pool free ratio is acceptable unless other above mentioned factors like Library Cache Hit Rate or Shared Pool Reload Ratio are not affected.The SQL statement to get the shared pool free ratio is as follows:

***SELECT***

***round((sum(decode(name,'free memory',bytes,0))/sum(bytes))\*100,2) "SHARED POOL FREE RATIO"***

***FROM***

***v$SGASTAT;***

Depending on the value of the shared pool free ratio, the value of **SHARED\_POOL\_SIZE parameter must be adjusted in the init.ora file.**

Apart from these parameters, one should also monitor the Library Cache Get Hits and Library Cache Pin Hits and ensure that these rates do not go below 95%

Thus, following guidelines can be suggested to tune the SHARED\_POOL\_SIZE parameter:

* If the shared pool reloads ratio increases more than 1 then SHARED\_POOL\_SIZE parameter must be increase. Also to make the most of the additional memory available to the shared SQL areas, number of cursors permitted per session must be increased. This can be achieved by increasing the value of the initialization parameter OPEN\_CURSORS.
* If the Library cache miss rate exceeds 1% or if the overall data dictionary cache miss ratio exceeds 1%, then SHARED\_POOL\_SIZE parameter must be increased.

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. | Parameter Name | Optimum Value | init.ora parameter to be modified |
| 1. | Shared Pool Reloads Ratio | Low | SHARED\_POOL\_SIZE,OPEN\_CURSORS |
| 2. | Library Cache Hit Rate | High | SHARED\_POOL\_SIZE,OPEN\_CURSORS |
| 3. | Data Dictionary Cache Hit Rate | High | SHARED\_POOL\_SIZE |
| 4. | Shared Pool Free Ratio | Small but non-zero | SHARED\_POOL\_SIZE |
| 5. | Library Cache Get Hit Rate | High | SHARED\_POOL\_SIZE,OPEN\_CURSORS |
| 6. | Library Cache Pin Hit Rate | High | SHARED\_POOL\_SIZE,OPEN\_CURSORS |

## REDO LOG BUFFER

Redo Log Buffer is a part of System Global Area (SGA) wherein changes made to a database instance are persisted. This enables the Oracle database to recover quickly in case of any media failures. Usually there are two or more pre-allocated files that store all updates made to the database, and each database instance is associated with one of these files. The database initialization parameter log\_bufferdefines the default size of the redo log buffer within Oracle 11g. Proper sizing of redo logs can greatly enhance performance. In our dashboard, we monitor the following three Redo Log parameters:

### Redo Space Wait Ratio

### When the redo buffers are small, transactions need to wait till space becomes available and they are recorded into the redo logs. The time for which the oracle log writer waits is reflected to the user, and is indicative of the fact that redo log buffer size is too small compared to the rate of transactions. Hence it is more prudent to increase the buffer size in such circumstances. Below query is used to calculate the Redo Space Wait Time Ratio:

***select round ((req.value/wrt.value)\*100,2)***

***fromv$sysstatreq, v$sysstatwrt***

***where req.name = ‘ redo log space requests’***

***and wrt.name= ’ redo writes’;***

### Redo Log Buffer Entry Ratio

### It is essential to monitor the Redo Log Buffer entries, for a positive value indicates that the buffer is full and the log writer is waiting to write the transactions. As these entries are crucial for failure recovery, hence they must be maintained in redo logs and the wait time should be negligible as it impacts performance. Below query is used to determine the Redo Log Buffer Entry Ratio:

***SELECT (a.value/b.value) "redo buffer entries ratio"***

***FROM v$sysstat a, v$sysstat b***

***WHERE a.name = 'redo buffer allocation entries'***

***AND b.name = 'redo entries'***

## REDO LOG FILES

Redo log files contain a record of all Data Manipulation Language (DML) commands carried out on the database. Usually these files are created when the database is created, but they can be added and deleted dynamically using Data Definition Language (DDL) statements. The number and size of the Redo log files is an important performance parameter. If the log files are too small or if there are not enough of them in the ring, Oracle may have to stall on a log switch.

Norms direct that there should be at least four redo log files, each 2 MB in size. A smaller number of Redo log files or a smaller Redo log file size can cause I/O bottlenecks. If the disk space is available, using more than four Redo log files further reduces the chances of delayed log switches. V$LOG view holds the current size of the redo log members. DBA monitor the number of log switches per hour, and if this number is too high, then the size of log files can be increased in order to bring it down to the recommended one switch per 15 to 30 minutes.

Redo Log Switch: A log switch occurs when the log writer stops writing to one redo log group and switches to another because the former ran out of space. By default log switch occurs after every 15 to 30 minutes. The system admin can also force a log switch if need be, for e.g. if the currently active group needs to be archived before it is completely filled. Following query has been used in our application to record the number of log switches that occur over a period of time:

***select to\_char(first\_time,'yyyy-mm-dd') day,to\_char(first\_time,'hh24') hour,count(\*) total***

***from v$log\_history***

***group by to\_char(first\_time,'yyyy-mm-dd'),to\_char(first\_time,'hh24')***

***order by to\_char(first\_time,'yyyy-mm-dd'),to\_char(first\_time,'hh24') asc***

## ROLLBACK SEGMENTS:

A rollback segment holds the old values (whether or not committed) of data that were altered by a transaction. Each database contains one or more rollback segments depending on the system requirements that used to provide read consistency, rollback transactions or to recover from system failures. Generally a rollback entry holds the block information, i.e. the file number and block ID corresponding to the data that was changed by a transaction as well as the data itself before it was changes. Oracle automatically links rollback entries for the same transaction, so that they can be located easily if required.

When we commit a transaction, Oracle empties the transaction history of that particular transaction in the rollback segment. If however, the user rollbacks this transaction, all the related data in the rollback segment would be applied back to the original tablespace, invalidating the changes made by the transaction.

In order to provide read consistency, rollback segments store *pre-image* of a data. When concurrent transactions are updating the database, the changes made to the data by one of the transactions should not be visible to others unless they are committed. This is especially important in case of long running queries.

The number and size of rollback segments is an important performance parameter. We examine the GETS and WAITS columns of the dynamic views v$rollstat and v$rollname in order to determine the rollback segment usage estimate its optimum size for our applications. GETS parameter gives the number of rollback segment header requests, whereas the WAITS tell us the number of header requests that resulted in waits. If the *waits* to *gets* ratio is greater than 0.01, then either new rollback segments should be created or the size of existing segments should be increased. We examine this ration using the below query:

***select round(sum(waits)/sum(gets)\*100,2) “HIT RATIO"***

***from v$rollstat";***

Rollback contention occurs when too many transactions try to access the same rollback segment concurrently. Consequently some of the transactions end up waiting. In order to avoid rollback contention, on an average one rollback segment should be created for every 4 concurrent transactions. We use the below query to check if there rollback contention exists in our application.

***select class, count***

***from v$waitstat***

***where class in('free list','system undo header','system undo block','undo header','undo block')***

***group by class,count;***

More rollback segments need to be added of the wait percentage is higher than 1 %.

## MEMORY AREA USED FOR SORTING

When an Oracle process performs a memory intensive operation such as a sort, hash join or group by it goes to the shared tunable memory area within pga\_aggregate\_target region and attempts to obtain enough continuous memory structures to perform the operation. This shared tunable memory part can usually be increased or decreased according to the system load.

If a client process is able to acquire these memory structures immediately, it is marked as an optimal memory access. If the memory acquisition requires a single pass through pga\_aggregate\_target, the memory allocation is marked as one pass. If all memory is in use, Oracle may have to make multiple passes through pga\_aggregate\_target to acquire the memory. Multipass executions are an indication of memory shortage. It is important to ensure that at least 95 percent of connected tasks can acquire their memory optimally. Following queries have been used to determine pga\_aggregate\_target parameter statistics stored in the dynamic views v$pgastat and v$pga\_target\_advice.

***select name,value from v$sysstat where name like 'workarea executions%';***

***select name,value from v$pgastat where name = 'aggregate PGA target parameter';***

***select round(PGA\_TARGET\_FOR\_ESTIMATE/1024/1024) as target\_size\_MB,bytes\_processed,ESTD\_EXTRA\_BYTES\_RW as est\_rw\_extra\_bytes,***

***ESTD\_PGA\_CACHE\_HIT\_PERCENTAGE as est\_hit\_pct,ESTD\_OVERALLOC\_COUNT as est\_overalloc from v$pga\_target\_advice;***

Depending on the optimal values of one pass and multi pass value and cache hit percentage of PGA\_TARGET, DBA changes the **SORT\_AREA\_SIZE** parameter in init.ora.

# ADDITIONAL PARAMETERS

We also analyze few additional parameters that can help DBAs further enhance the database performance. They are briefly discussed as under:

1. Short/Total Table Scans: Creation of index on tables usually facilitate the table scans, however this might not be the case if the tables are short. Oracle can perform table scans on shorter tables far more quickly without using an index. Below query is used to determine the proportion of full table scans which are occurring on short tables.

***select round((shrt.value/(shrt.value+lng.value))\*100,2)***

***from v$sysstat shrt, v$sysstat lng***

***where shrt.name='table scans (short tables)' and lng.name='table scans (long tables)';***

Ideally this value should be high. Lower figures may indicate lack of indexes on large tables or poorly written SQL which fails to use existing indexes or is using the indexes having high selectivity factor.

1. Table Contention Ratio: This metric indicates how well table storage is working. Non-optimization of this ration can lead to chained row continuation, which occurs either due to row migration of row chaining. These issues can be handled by increasing the value of PCTFREE factor. Another issue arises when multiple processes try to insert data in a single table, known as free list contention. The table header structure maintains one or more lists of blocks which have free space for insertion. If more processes are attempting to make insert than there are free lists some will have to wait for access to a free list.

Following two figures have been used to determine the Table Contention Ratio in our application:

***--Chained Fetch Ratio***

***select round((cont.value/(scn.value+rid.value))\*100,2) from v$sysstat cont, v$sysstat scn, v$sysstat rid where cont.name= 'table fetch continued row'***

***and scn.name= 'table scan rows gotten' and rid.name= 'table fetch by rowid';***

***-- Free List Contention***

***select round((sum(decode(w.class,'free list',count,0))/(sum(decode(name,'db block gets',value,0))+ sum(decode(name,'consistent gets',value,0))))\*100,2)***

***from v$waitstat w, v$sysstat;***

1. CPU Parse Overload: The CPU parse overhead is the proportion of database CPU time being spent in parsing SQL and PL/SQL code. High values of this figure indicate that either a large amount of once-only code is being used by the database or that the shared sql area is too small. The parameter SORT\_AREA\_SIZE needs to be altered in init.ora if the CPU Parse Overload is too high.

***select round((prs.value/(prs.value+exe.value))\*100,2)***

***from v$sysstat prs, v$sysstat exe***

***where prs.name like 'parse count (hard)' and exe.name= 'execute count';***

### Wait Times:

* The presence of a larger number of workers is not always advantageous - there are times when the workers step on each others' toes to acquire locks (both internal / external) and sessions wait for each other while showing up as 'running'.
* This also occurs when a previous 'runaway' session holds a lock and the worker is waiting on this hung session to complete.

A simple method of monitoring this is to use the following query:

***select sid,event,state,seconds\_in\_wait from v$session\_wait;***

***select event, count(\*) from v$session\_wait group by event;***

# On-demand Reports

Our application continuously monitors the database and collects the values of some of the above mentioned performance metrics on an hourly as well as periodic basis. It also allows the user to request these statistics for a specific duration, i.e. the DBA and key in the start date/time and end date/time for which he needs monitor the performance parameters. In response to his request, a report is generated in pdf format and presented to him. A sample report generated by our application is provided below.



Oracle also supports automatic generation of the report of performance statistics through **Automatic Workload Repository (AWR)**. Our application also supports few AWR functionalities, namely management of snapshots and baselines. The AWR automatically collects performance statistics at regular intervals and displays the report in text or html format. These can be object statistics, that refer to the access and usage statistics of the database segments or they can be time model statistics that represent the utilization time of various activities, or simply the execution plan of a sql query. Collection of these statistics by AWR is enabled by default, and can is controlled by **STATISTICS\_LEVEL**parameter, which must be set to *TYPICAL* or *ALL* (default setting being TYPICAL). The AWR scripts are located in *$ORACLE\_HOME/rdbms/admin* directory.

AWR collects the performance statistics at regular intervals and the result is called an *AWRSnapshot*. The default configuration is set to half hour intervals and seven days of history. In our project we reset the snapshot collection interval to the minimum possible 15 minutes to permit users to zoom in and view the performance of various parameters at a finer granularity. The user can also pull up an entire HTML report for a specific time interval, and this functionality is supported by the *baseline* feature of AWR.A statistical baseline basically a collection of statistics when the system is running at peak load. Comparing these statistics with the ones persisted during normal circumstances enable the DBA to discover parameters that have deviated significantly and could be the cause of the problem.

The complete set of AWR related queries and procedures used in this project can be found in *awr.sql*.

# DEBUG INTERFACE

Our application allows the DBA to issue SQL commands to the database, and the query results are displayed neatly on the screen.

# SCREENSHOTS

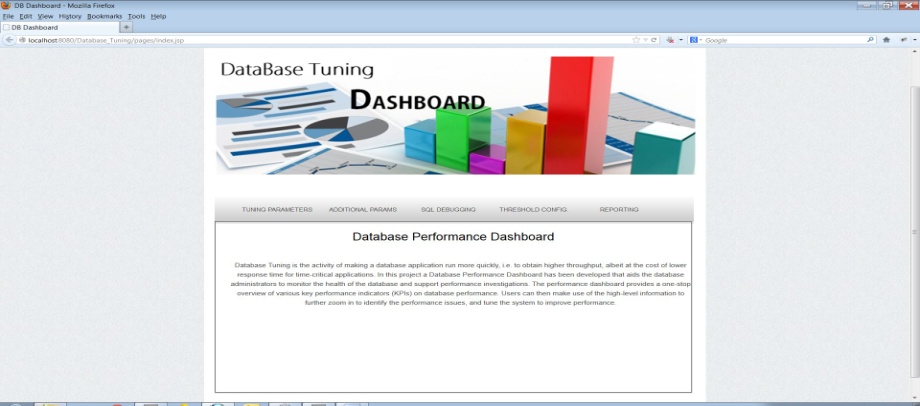


Fig 1: Database Performance Dashboard Homepage



Fig 2: Gauge meter displaying the Data Dictionary Cache Hit Ratio statistic



Fig 3: Gauge meter displaying the Library Cache Hit Ratio statistic



Fig 4: Gauge meter displaying the aggregate value of Dictionary Hit Rate on an hourly basis

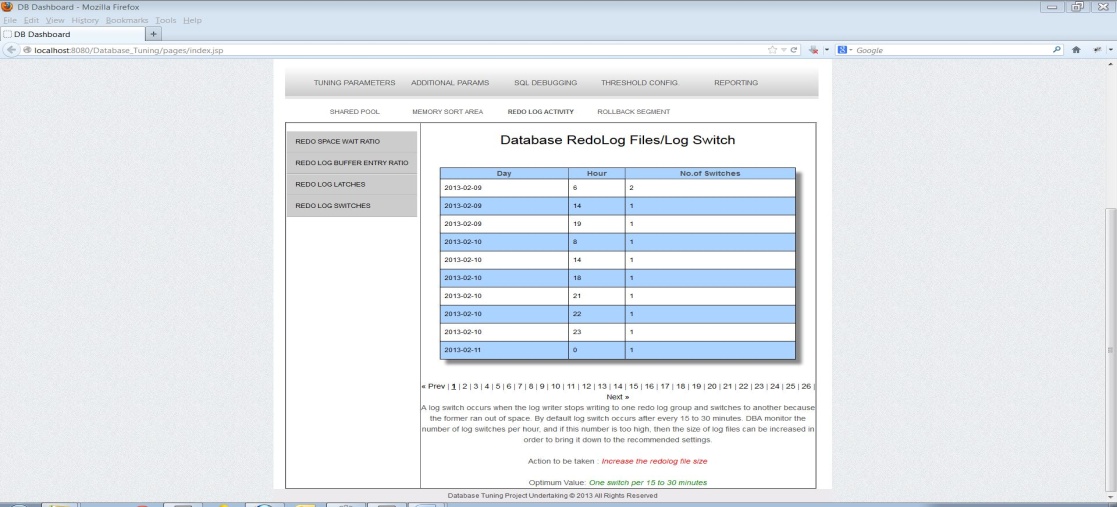


Fig 5: Table showing the statistics of Redo Log Switch

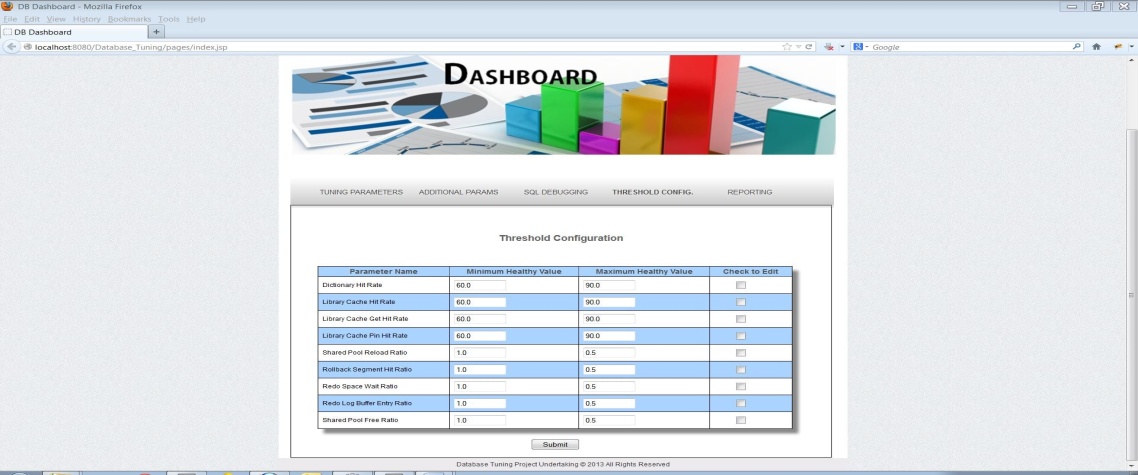


Fig 6: Configuration Page for various performance parameters

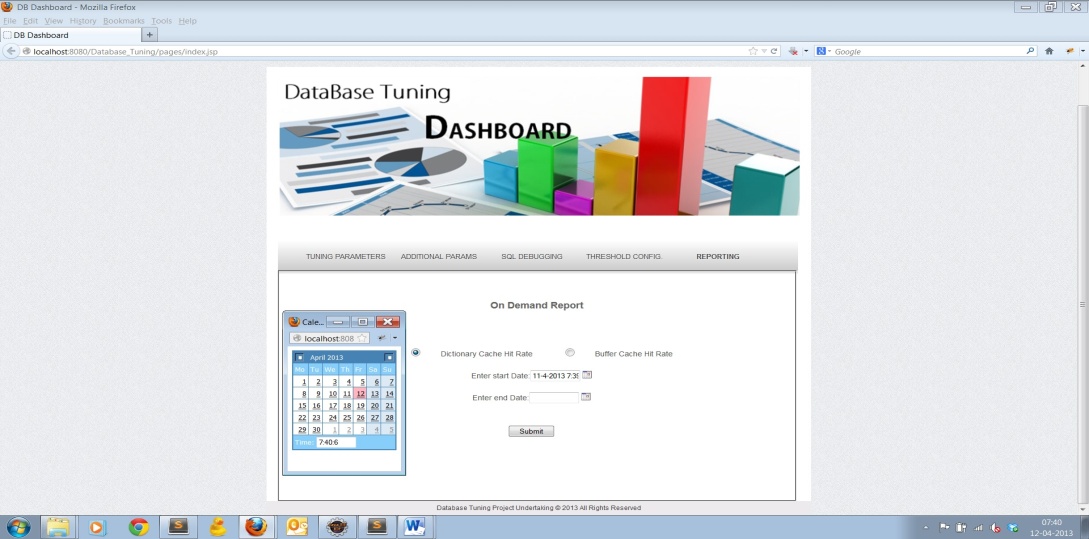


Fig 7: On-demand Report generation page

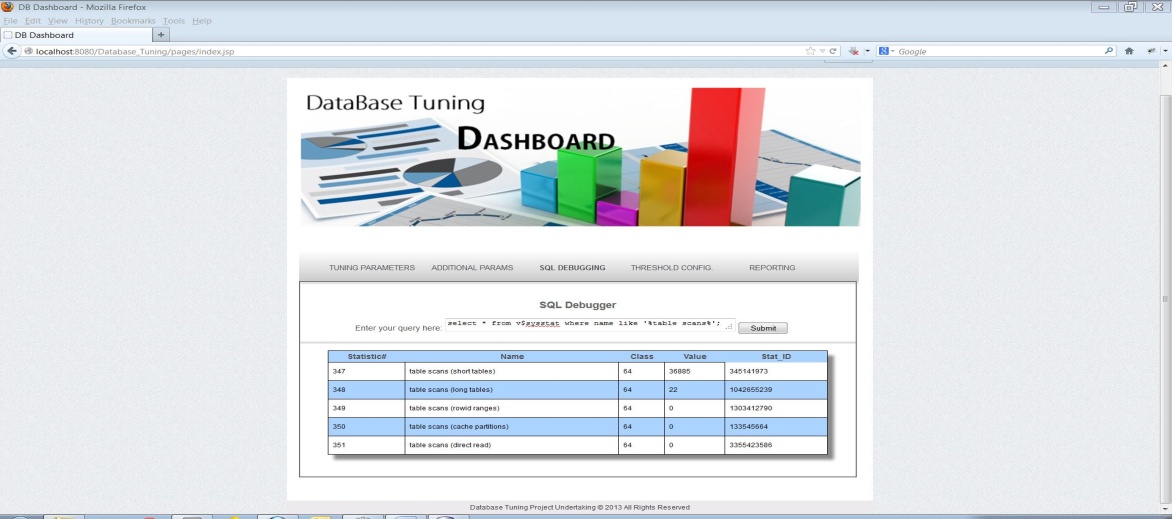


Fig 8: SQL Debug Interface

# CONCLUSION

The goal of this Dashboard application was to monitor the Inventory Management Database from a DBA perspective, and take necessary measures to tune the database and enhance performance. We picked up some vital tuning insights during the course of this project, and we summarize them as under:

* Oracle maintains numerous performance statistics in various dynamic views like v$sesstat, v$syssstat etc which enables DBAs to monitor the health of these parameters.
* Tuning the buffer cache is essential to prevent increased disk access which adversely affects the performance of the database
* The System Global Area (SGA) forms the part of the system memory (RAM) shared by all the processes belonging to a single Oracle database instance. The SGA contains all information necessary for the instance operation, such as data dictionary cache, buffer cache, library cache etc.
* Estimating SGA size is critical for all applications as it ensures that RAM is neither under-allocated nor over-allocated.
* Redo log buffer and files play a critical role in recovering the oracle database in-case of media failure. Hence it is important to maintain sufficient buffer space and file sizes and minimize log switch in order to optimize performance. Rollback segments hold the pre-image of data that were altered by a transaction and serve three main purposes: to rollback changes made to the database before commit, provide read consistency, and to restore system state in case of failures. The number and size of rollback segments is an important performance parameter.
* AWR automatically generates a report of all relevant performance statistics periodically that greatly aids the DBA in tuning the database.

# REFERENCES

1. <http://docs.oracle.com/cd/B28359_01/server.111/b28274/autostat.htm#CHDHBGJD>
2. <http://docs.oracle.com/cd/B19306_01/appdev.102/b14258/d_wkrpos.htm>
3. <http://www.dbspecialists.com/files/presentations/buffercache.html>
4. <http://www.toadworld.com/KNOWLEDGE/KnowledgeXpertforOracle/tabid/648/TopicID/MBUP2/Default.aspx>
5. <http://rohanrajkalra.blogspot.sg/2008/11/tuning-pgaaggregatetarget-in-oracle.html>
6. <http://rohanrajkalra.blogspot.sg/2008/11/tuning-pgaaggregatetarget-in-oracle.html>
7. <http://www.hoopoes.com/cs/oracle_tune.shtml#lhit>
8. <http://www.dba-oracle.com/concepts/shared_pool_concepts.htm>
9. <http://www.dba-oracle.com/m_library_cache_hit_ratio.htm>