Moving to a Proactive Database Environment

Lets face it, managing a Database environment can be very difficult to do. Who amongst you as DBAs have constantly found yourselves reacting to each situation that occurs ? The table runs out of extents right near budget time, and when fixing it you find there is no room left on the disk to expand the tablespace the table is in. By now you are in panic mode, trying to find some where to put the datafile, but there isn't any room left, anywhere. Why didn't we buy some more disk when we had the chance, you ask yourself ? That's right, you never had time to request it because you were to busy fixing problems.

As a database administrator you will find that you are constantly fire fighting, sometimes controlling the blaze but never putting it out. Only by moving to a proactive environment can you overcome the burdens and inefficiencies of the reactive environment you might be in and in doing so enter an optimally controlled and managed one. Such an environment offers many benefits including a reduction in database downtime, a finely tuned database and an improvement in productivity.

So why the need to always react ? The answer is found in the work practises of the environment. By taking a step back and then looking at how you are doing your work is the first step, but more on this later. Lets first have a look at why this situation has occurred.

With the need to cutback on resources and increase productivity, the workload of the DBA can be cut first because it is seen as not directly benefiting the client. With the resources cut right back, only activities which are seen to be important can be focused on.

Lets have a look at two typical scenarios that are most likely to be encountered :

1. The users of an application have complained bitterly to management about the slow performance of it. Management are now asking you to drop everything and fix it as a matter of high urgency. This means spending a large amount of time tracking the problem down. This involves looking to see if there is a problem with the tuning of the application, a problem with the tuning of the database or if there is insufficient machine capacity. In all cases it is up to you, to find where the fault is, and this takes time.
2. A table modification change is urgently required in the production database. The users are in desperate need of it. As the DBA, you were just told to implement the change and take for granted that it will not impact the database. As it is urgent, there is no time to properly review what is happening in the upgrade. The next day the discovery is made that the upgrade has forced some tables to go into over 50 extents and there is a critical shortage of free space left and this now has to be addressed.

In both cases time and effort is being spent doing extra work which could have been prevented with the right amount of planning.

So to do your job properly, requires a fundamental change in how you do your work, with the aim to become proactive. The goals are simple but can be difficult to implement :

- Anticipate and prevent problems before they occur.
- Optimally tune the database.
- Optimally manage storage.
- Optimally tune the network.
- Minimise impact to the database.

There are numerous following such goals. Optimally managed resources ensures that the environment is efficiently tuned and managed. Also, an optimally managed environment ensures your resources are used efficiently and cost effectively.

Minimising impact to the database involves reducing the amount of maintenance that is applied to the database. This in turn will ensure that there is an increase in uptime for the database and reduce the risk of an error, especially an error resulting from fatigue caused by working on the database at odd hours of the night.

Role of the DBA

The remaining part of this article will detail how such a move to a proactive environment can be achieved. Moving to a proactive environment is easier said than done. There are a number of hurdles that must be jumped, but when reached the benefits are worth the effort.

To start, a complete change in the philosophy for how a database is managed is needed. The role of the DBA needs to be revised. By focusing on such a new role, the move to a proactive environment will be so much easier.

The new role of the DBA can be stated as :

'*To ensure that the database performs to its optimal, is fully secured and can be recovered in time of need.'*

To achieve this, the DBA can no longer be consigned to the back room, out of sight and out of mind. The DBA has to become more actively involved.

Ensuring optimal performance

The first part, *'ensure that the database performs to its optimal'*, is the most difficult to implement and involves a number of steps.

For starters, one has to throw out the window the concept that performance tuning is an action which is done after the application is built. In the database of the nineties, application and database tuning go hand in hand, and must be factored in from the very beginning.

There are three critical inputs which must be analysed when an application is built. They are the user interface, performance and the database. Each interacts with the other and each has *equal* weighting (see Figure 1).

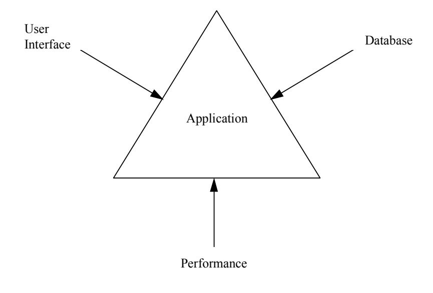

Figure 1 : The three most critical inputs for application development, from the DBA viewpoint

Only with the advent of GUI application programming has the issue of the user interface become apparent. An efficient GUI design means that the application is efficiently used resulting in a reduction in database and network calls.

The design of the database must take into account performance, and the user interface will also affect the design.

All three inputs into the application design require co-ordination by the DBA. They have the knowledge on how the application works in the environment and are in the best position to control how the application integrates with the database.

Cyclic Maintenance

To ensure that the database environment is optimally tuned, the move must be away from reacting to events, and instead actively plan and then tune the database. There is a balance between constantly performing maintenance on the database and not interfering with the database. At times it is true when it is said that problems occur only after the maintenance has been done on the database. So a good period to aim for where no work is done on the database is about six months (depending on the volatility of a database application, the period can range from 4-8 months).

Maintenance is introduced and performed on a cyclic basis. The cycle involves reviewing the database, performing maintenance, and then leaving the database alone (see Figure 2). A once off review is performed two to three months after the database has been created, as newly created databases are prone to change.

The aim is to perform a full database reorganisation and tune every six months. Performing this maintenance more frequently will be disruptive to the users and unnecessary. Performing it less frequently will result in the database becoming out of tune and the danger that objects will grow beyond their original storage allocations. If for example, a two year period was used, then it would be quite difficult to predict storage requirements. There are too many factors to take into consideration and uncertainties that can occur.

Once the maintenance has been performed the database is not touched except if an emergency occurs. So the role of the DBA changes further and they must now become an expert in forecasting to calculate storage and CPU requirements for a six month period.

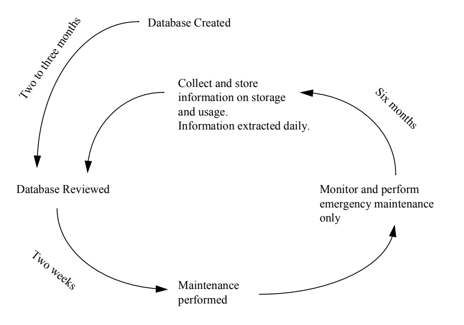
Database Created
Two to three months
Collect and store information on storage and usage. Information extracted daily.
Six months
Database Reviewed
Monitor and perform emergency maintenance only
Two weeks
Maintenance performed

Figure 2 : Maintenance Cycle

Emergency maintenance is only performed when something drastic occurs, that is the stability of the database is *threatened.* Examples include a table not being able to grow into its next extent or the PROCESS parameter being exceeded in the INIT.ORA file. In these cases emergency maintenance has to be performed because an event unforseen in the initial planning was missed. These events happen but should occur rarely. If they occur frequently then the review has not been performed correctly and procedures should be adjusted accordingly.

Database Review

The review of the database is the most critical step, and approximately two working weeks should be devoted to it. The review involves :

- Analysing information collected about the database from the previous six months and forecasting growth and database usage.
- Liaise with areas (see Table 3) and determining potential changes to the environment in the next six months. For example, there might be a plan to double the number of users who access the database.
- Liaising with management to acquire extra storage and capacity based on forecasts. If due to cost constraints, this capacity cannot be acquired, then alternatives must be explored.

The key to the review is obtaining information. This is best handled by the DBA coding plus running scripts and then storing information about all the objects in the database. The database is the best environment for the DBA and PL/SQL the best tool. The information extracted can be broken up into coarse and fine grain (see Table 4).

Developers	•Determine upgrades planned in the next 6 months •Review indexes and SQL statements
Application Users	•Review application usage •Review data entry usage
Application Management	•Determine application changes in the next 6 months •Determine changes to capacity in the next 6 months
System Administrators	•Determine operating system changes planned in the next 6 months •Review capacity changes required in the next 6 months
Storage Management	•Determine if there are any hardware changes in the next 6 months •Review storage requirements for the next 6 months
Communications	• Determine if there are any communication changes in the next 6 months • Review capacity requirements for the next 6 months

Table 3 : Areas the DBA should liaise with when performing a review

The initial investment required in moving to a proactive environment is for the time to devote to building the scripts and programs required to extract the information from the database and then store it (See Inset). It is this hurdle which is the hardest one to jump, as it is typically seen as a waste of time and effort. Unfortunately, there are no known tools on the market which perform all of this for you.

It is important that information should be extracted on a daily basis and stored in a central repository (see figure 5). This repository is rather like a data warehouse. Information extracted from the database is used for two purposes. The first, as discussed already, is used for the six monthly review. The second purpose is to test to see if emergency maintenance is required.

Coarse Grain	Fine Grain
Tablespaces Datafiles Database statistics - UTLSTAT Placement of files in the disk structure INIT.ORA parameters Network load	Tables - Storage - Size Indexes - Storage - B Tree stats Rollback Segments

Table 4 : Information required to perform analysis

It is important that the emergency maintenance report details only objects that need to be fixed immediately. The danger, which is common in a lot of environments, is information overload. By presenting too much information, the odds increase of vital pieces being overlooked and missed.

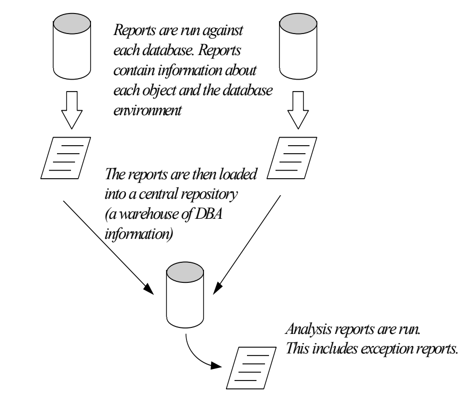
Reports are run against each database. Reports contain information about each object and the database environment
The reports are then loaded into a central repository (a warehouse of DBA information)
Analysis reports are run. This includes exception reports.

Figure 5 : Creating a DBA warehouse

Forecasting

Forecasting growth can be performed automatically using linear analysis provided one basic assumption is made and that is that growth on the table is constant. By monitoring growth over a period of time it becomes possible to fit a straight line to it and then predict if a table will exceed its storage allocation (see Figure 6 and Table 7).

Extracting the data from each database is best performed using PL/SQL. A procedure is run which collects all the information, summarises it and stores this information into a temporary table. This table is then exported or unloaded (using SQL*Plus) to an operating system file. It is then loaded into the central DBA repository. Alternatively the data can be copied over using SQL*Net. Table 8, shows examples of two PL/SQL procedures that can be used to extract the information from the database and the table used to store this information.

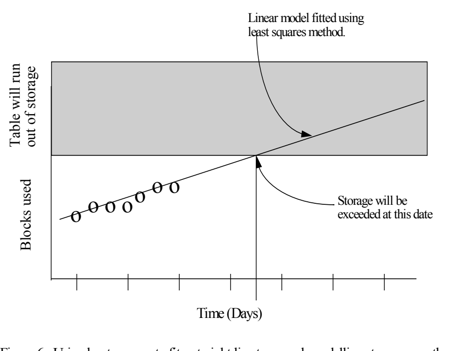

Figure 6 : Using least squares to fit a straight line to a graph modelling storage growth for a hypothetical table. Growth is assummed to be constant.

```
/* This script will fit a straight to a set of points using the Method of Least
   Squares. The table being analysed has been simplified, and contains two
   columns X and Y. The X column relates to Julian days. The Y column relates to
   the size of the table in blocks. The report will extend the line &1 days,
   and test to see if there is sufficient storage in the tablespace (max_free)
   to store the information.This code can be simplified if 7.1 embedded functions
   are used. */

SELECT owner || '.' || table_name,
       (trunc ( avg(y) -
        (((( count(x) * sum(x * y)) - (sum(x) * sum(y)) ) /
          (( count(x) * sum(power(x,2))) - power(sum(x),2))) * avg(x))
         +
         ((( count(x) * sum(x * Y)) - (sum(x) * sum(y)) ) /
          (( count(x) * sum(power(x,2))) - power(sum(x),2)) *
          to_number(to_char(sysdate + &1, 'J')) )) ),
       max_free
FROM     table_x_y
GROUP BY owner, table_name
HAVING
       ( count(x) * sum(power(x,2)) - power(sum(x),2) <> 0)
 AND
   /* Test if (forecast growth - currently used) > storage free */

       (trunc ( avg(y) -
        (((( count(x) * sum(x * y)) - (sum(x) * sum(y)) ) /
          (( count(x) * sum(power(x,2))) - power(sum(x),2))) * avg(x))
        +
         ((( count(x) * sum(x * Y)) - (sum(x) * sum(y)) ) /
          (( count(x) * sum(power(x,2))) - power(sum(x),2)) *
          to_number(to_char(sysdate + &1, 'J')) )) )
        -
         blocks_used  >  max_free;
```

Table 7 : SQL Code used for forecasting growth (all code not shown)

Once all the information has been collated, use the following checklist for doing maintenance on the database :

- Recreate all tables with enough storage for 6 months growth and store the table in one extent. Downsize tables if necessary.
- Allow room in the tablespace for five extents worth of unanticipated growth, each extent size to be consistent (see Table 9).
- Recreate all indexes (storage as for tables)
- Recreate all tablespaces. Defragment them and store them in one datafile, except if coarse grain striping is being used (see Table 10).
- Remove obsolete users and objects.
- Modify INIT.ORA parameters.
- If the cost based optimiser is being used, then run full statistics on all objects (or estimate if you are satisfied with the results).
- Review security.

By following the above steps, and doing a thorough review of the database, satisfaction can be gained that the database is correctly tuned, and will stay tuned for the period of 6 months. This will leave more time for you as the DBA to perform important tasks like reviewing SQL code created by developers and ensuring this code is accessing the database optimally.

Securing the Database

The next step is to ensure that *'the database is fully secured'*. This requires another change in philosophy. The guiding premise is :

'The DBA owns the objects in the database, and is responsible for them. The DBA does not own the data in each object. This responsibility is left to an application manager.'

By owning each object in the database, the responsibility for ensuring that each object is backed up and can be recovered is firmly entrenched in the hands of the DBA. They also becomes responsible for ensuring each object has the correct security on it, and has sufficient storage. Other responsibilities include the management of indexes, constraints, synonyms, object tuning and database links.

Initial	Next
10	10
20	10
50	10
100	20
200	50
500	100
1000	100
2000	200
5000	500
10000	500
20000	500
50000	500
... etc	... etc

This chart details initial and next extents to be assigned to database objects. It is important to stick to figures which can be easily absorbed by the DBA. This reduces the chance of errors occurring when interpreting figures, and improves on the understanding of how storage is being managed inside the database. Once the next extent size reaches 500, it stays constant. This simplifies storage management, as extents are now small and controllable. Fragmentation is less likely to occur as database extents are consistent in their size.

Table 9 : Next Extent sizing chart

What the contents of each table are is not of importance. How an application works and hangs together is the responsibility of the developers and the application manager.

From this premise the following work principles can be determined :

- The DBA is not allowed to run scripts on behalf of developers which manipulate data in tables. As the DBA has no knowledge about the contents of data in tables, they are not in a good position to determine if the scripts are valid. There is also the potential for a security breach to occur.
- Developers should not run scripts in a production database which will modify the structure of tables.
- The DBA has a right to know how tables are being manipulated, ie. what SQL statements are run against database tables.

In addition to the above security controls, the security of the database should be reviewed when the 6 monthly maintenance review is being performed including :

- Ensure all database users point to the correct default and temporary tablespaces.
- Check all users with DBA type privileges, and make sure they are valid.
- Check operating system permission on all datafiles.
- Redundant accounts are removed.
- Roles and grants are valid and are pointing to the correct objects, privileges and users.

Data Recovery

The final step in moving to a proactive environment is, *'to ensure that the database can be recovered in time of need'*.

- Keep tablespaces to 100 Meg or less
    - unless object is greater than 100 Meg
    - easier to manage
- Do not have more than 10% free space per tablespace
    - unless growth patterns warrant this
- Keep frequently accessed objects in separate tablespaces
    - balance I/O
- Keep each tablespace in a single file
    - unless using coarse grain striping
    - reduces disk head movement when searching
    - use O/S features for fine grain striping
- Locate index and tables in separate tablespaces
- Set default initial and next extent values to be twice the size of the tablespace.
    - prevents accidental table creation
    - enforces table creation scripts to include an Initial and Next extent storage clause on them

Table 10 : Tablespace management tips

To achieve this, an actual recovery of the production database must be performed at least once a year to check that :

1. The backups are working correctly.
2. There are storage devices available that can be used to recover the database.
3. There is sufficient knowledge and expertise in recovering the database.

The backup strategy should be reviewed and the following questions should be asked :

1. If the database increases in size, can the backups cope ?
2. Will the size of export files grow ? If so, is there sufficient storage to contain them ?
3. Will the length of time a backup take to run increase ?
4. Is there sufficient disk storage to handle an increase in the size of the database ?

The ability to perform a recovery includes testing the following scenarios :

1. A datafile is lost
2. The redo logs are lost (and mirroring is not activated)
3. The latest backup failed, and recovery has to be performed from an older backup.

Once the move has been made into a proactive environment, discipline is required to ensure that the environment remains stable. This means that regular database reviews have to be performed, security enforced and recovery procedures tested. It is very easy to slip and move back into a reactive environment.

So, the encouragement is there to move to a proactive database environment. Such an environment offers a lot of advantages, including an increase in database uptime, minimising the chance of problems and errors occurring, finding problems quickly and also and improvement in productivity.

It is not easy to move to such an environment and when reached requires discipline to maintain it. Once reached though, the benefits are many and should offer you greater control and flexibility in managing the database.

Article produced by Peter Sharman and Marcel Kratochvil.
This article was created whilst Peter was working for the Department of Human Services and Health (Australia).
Marcel is currently working for Oracle as a consultant in Canberra, Federal Government Division.

Inset : **The issues of packaged procedures**

Planning on writing some PL/SQL for your Oracle database ? If the answer is yes, then you are not the first person to be doing this. With release 7.1 of the Oracle database a wonderful new feature of being able to embed your own function in a SQL statement was introduced.

Such a feature will undoubtably open up a Pandora's box. Developers will start to write more code in PL/SQL. If these developers are in a client/server environment, then they should be putting all the business code into packages and storing this also in the database. So, rather than the current 4 or 5 Meg of PL/SQL being stored in the system tablespace, storage of over 100 Meg will be required, and possible a whole lot more.

So what are the issues ?

- What is performance like if over 100 Meg of PL/SQL is stored ?
- How do you manage all this code ?
- How much memory is really required ?
- How do you know what code is being used and what code isn't ?
- How can you tell the performance of package ?
- Can packages be placed in other tablespaces ?
- What is efficient PL/SQL code ?
- Will core package modules be written so that common PL/SQL libraries can be shared amongst all users ?

The release of Designer/2000 first highlighted this issue. Over 50 Meg of extra storage was required for the system tablespace.

Very soon, third party vendors will start writing their applications in PL/SQL packages, increasing the storage requirements even further. And to really throw a spanner in the works, what about applications like Oracle Financials, how much storage will be required to hold all its business rules? 200 Meg ? Are we ready yet to manage such large amounts of code ?

Now is the time to start preparing, planning and investigating what the real issues are behind storing large amounts of PL/SQL code in the database, before it becomes a real issue. Now is the time to be proactive.

Table 8 : Example of PL/SQL code used to extract statistics about tables from the database

```
/* This table is used to store statistical information about a table
   in the database. Extensions to the table include adding a date column
   so that information can be stored historically and adding columns
   to store more information about storage. */

CREATE TABLE table_statistics
 (
  owner                        varchar2(20),
  table_name                   varchar2(40),
  extents           number,
  max_extents                  number,
  initial_extent    number,
  next_extent                  number,
  used_blocks                  number,
  blocks_allocated  number,
  tablespace_name   varchar2(20),
  tablespace_free   number,
  total_rows                   number);

/* Note : This code can only be run against a V7.1 or greater Oracle database */

CREATE or REPLACE PROCEDURE
 ind_table_statistics (v_owner IN VARCHAR2, v_table_name IN VARCHAR2)
AS
  v_extents         number;
  v_max_extents                number;
  v_initial_extent  number;
  v_next_extent                number;
  v_used_blocks                number;
  v_blocks_allocated           number;
  v_tablespace_name varchar2(20);
  v_tablespace_free number;
  v_total_rows                 number;

  source_cursor         integer;
  ignore                integer;

BEGIN

 /* This section of code involves creating and then running a dynamic SQL
    statement. The aim of the SQL statement is to count the total number
    of rows and used blocks in a table. This information can be extracted
    without impacting the statistics collected by the cost based optimiser.
    Note : The Oracle user running this procedure must have been granted SELECT
           access on the table. */

 source_cursor := dbms_sql.open_cursor;
 dbms_sql.parse(source_cursor,
  'select nvl(count( distinct substr(rowid,1,8)), 0),
       nvl(count( rowid ), 0) from ' ||
  v_owner || '.' || v_table_name,
 dbms_sql.v7);

 dbms_sql.define_column(source_cursor, 1, v_used_blocks);
 dbms_sql.define_column(source_cursor, 2, v_total_rows);
 ignore := dbms_sql.execute(source_cursor);

 IF dbms_sql.fetch_rows(source_cursor) > 0 then
  dbms_sql.column_value(source_cursor, 1, v_used_blocks);
  dbms_sql.column_value(source_cursor, 2, v_total_rows);
 ELSE
  v_used_blocks := 0;
  v_total_rows  := 0;
 END IF;

 dbms_sql.close_cursor(source_cursor);

 /* Extract storage information about the table. */

 SELECT extents, max_extents, initial_extent, next_extent,
        blocks, tablespace_name
 INTO
        v_extents, v_max_extents, v_initial_extent, v_next_extent,
        v_blocks_allocated, v_tablespace_name
 FROM   sys.dba_segments
 WHERE  owner = v_owner and
        segment_name = v_table_name;

 /* Extract storage information about the tablespace the table is in */

 SELECT nvl(max(blocks), 0)
 INTO   v_tablespace_free
 FROM   sys.dba_free_space
 WHERE  tablespace_name = v_tablespace_name;

 INSERT INTO table_statistics
  (owner, table_name, extents, max_extents,
  initial_extent, next_extent, used_blocks, blocks_allocated,
  tablespace_name, tablespace_free, total_rows)
 VALUES
 (v_owner, v_table_name, v_extents, v_max_extents,
  v_initial_extent, v_next_extent, v_used_blocks, v_blocks_allocated,
  v_tablespace_name, v_tablespace_free, v_total_rows);

COMMIT;

END ind_table_statistics;
/

CREATE or REPLACE PROCEDURE
 all_table_statistics
AS

/* The purpose of this cursor is to fetch table names from the
   database that the user running this procedure has been
   specifically granted SELECT access on. Having DBA privileges
   is not sufficient. */

CURSOR c1 IS
SELECT  owner, table_name
FROM    user_tab_privs_recd
WHERE   privilege = 'SELECT' AND
        owner not in ('SYS'); /* Add other applications here */

BEGIN
 FOR c1rec IN c1 LOOP
  ind_table_statistics( c1rec.owner, c1rec.table_name);
 END LOOP;

END all_table_statistics;
/
```