Exploding the Myths

By
Marlene Theriault and Rachel Carmichael

About this paper

Each year for the East Coast Oracle (ECO) conference, we put together a workshop based on Oracle DBA topics. In 1997, we created a workshop called "DBA 101: A Refresher Course" which became a paper and then a book. For ECO'2000, we wanted to present some of the statements that we've heard that either sound true but are not, sound wrong but are actually correct, and some statements that may be correct or incorrect depending on your particular environment and requirements.

The workshop was a wonderful success and we've had the pleasure of presenting the material a few more times as a full-day workshop. As with "DBA 101", we feel that it is time to create a paper based on the workshop to be able to distribute the information on a broader scale more easily. This paper is the result of our efforts. We hope that you find some valuable "nuggets of information" within it.

We have divided the statements into several different categories and we request that you take a moment or two to think about each statement and decide if you think that it's true or false before you read our determination of its veracity. Above all else, we hope that you will enjoy this adventure into examining and, hopefully, exploding some well-worn myths.

Also, please keep in mind that, for the "older" DBAs who are reading this paper, there are many precepts that we follow because we found long ago that they work. However, with the newest technology, approaches that we are used to using are no longer valid and we may need to learn to embrace new paradigms to remain effective DBAs.

Dealing with Fragmentation and Extent Management

Fragmentation is a problem that DBAs have traditionally spent a good deal of time worrying about. Let’s take a look at some of the fragmentation issues and ways of dealing with them and see if these are myths or truths.

Set pctincrease to 1 in all tablespaces (True or False?)

On the surface, this statement seems to really make a lot of sense. The thinking behind it is: If the tablespaces all have **pctincrease** set to a non-zero number, then whenever SMON wakes up, it will coalesce adjacent free space. The idea behind setting the value to "1" is that any number multiplied by 1 should remain the same value and, therefore, there should be little or no impact on the size allocated for each next extent. However, the fallacy is that the value that is used is not 1 but actually 1.1 so there's an actual, steady increase of the next extent value and a "creeping up" of the size of the object's extents.

By default, when you create a tablespace and do not declare the values for a storage clause, Oracle uses a **pctincrease** value of 50. Many DBAs plan on setting the value to 0 for every object they create so that they can avoid extents that are increasing in size – either slowly or quickly. But the problem here is that if you forget just once to set the storage parameter on an object, you may CAUSE fragmentation, rather than relieve it.

By setting the **pctincrease** to a non-zero number, the thinking is that you will no longer have to worry about tablespace fragmentation… or at least not have to worry about whether or not free space is coalesced at a time when it will have a performance impact. But SMON wakes up and coalesces on its own schedule, not yours. And SMON only coalesces one tablespace at a time, and only 8 extents in that tablespace. So if you have a good deal of fragmentation, SMON is not going to clean it up quickly or when you want it to.

A better solution is to have 3 tablespaces with extent sizes of “small”, “medium” and “large” where the **initial** and **next extent** values in each of these tablespaces is the same, and where **pctincrease** is set to 0 on the tablespace level. Doing this means that you will never need to coalesce extents to get one of the right size since all of them, by default, are the same, correct size.

This is the idea behind the new minimum extent clause on tablespaces in 8i, where all extents that are allocated are multiples of this minimum number.

Size everything to the last K so you'll never have any extending in your tablespace (True or False?)

False… especially if you take to heart what we said previously about some extension not being a problem. Remember, too, our suggestion to use small, medium and large extents in separate tablespaces and place your objects in the appropriate tablespace.

Sizing to the last K can

1. Take most of your time
2. Add to fragmentation
3. Be wrong a good deal of the time

You can only size this closely if you have both accurate details on average sizes of columns and on growth rates of the tables.

In reality, you can spend your entire life trying to get the sizing right only to find that, over time and with usage changes, everything you did to determine precise sizes of objects has fallen by the wayside. The more you create, populate, and size databases, the more you will gain an instinct for ball park estimations of what the proper sizes for objects should be.

Recreate objects as soon as they have more than a few extents (True or False?)

False… Well, sort of. In earlier versions of Oracle, you had a hard limit on the number of extents you could have, based on the database block size (2k block size was 121 extents, 4k was 249, 8k was 505). With 7.3, Oracle added the ability to create an object with maxextents UNLIMITED (which is really 2GB or 2147483648 extents – more than you could ever conceivably need).

So you never need to recreate your objects right? Well, yes and no. If you have objects that you never ever delete from, then a really large number of extents is not a problem. But if you regularly insert and delete from your table, you will be extending the highwater mark and causing Oracle to have to scan more and more blocks when you do a full table scan.

When you delete data from blocks, and the blocks are emptied, Oracle adds a record to the FET$ (Free Extent) table and removes one from the UET$ (Used Extent) table for each extent that is freed. So if you are doing a lot of deleting from a table with a large number of extents, this overhead can impact performance. As an example, a friend of ours did an experiment – he did a **truncate table drop storage** on a table with 60,000 extents. It finished 16 hours later.

So the answer is, sometimes you want to recreate objects with a large number of extents. Oracle's current philosophy is to recommend you recreate the object if you have more than 4000 extents. We would suggest beginning to monitor, and plan for re-creation, when the number of extents is about 1000 depending on what version of the RDBMS you are running and how much space you have available.

Don't monitor space, turn autoextend on (True or False?)

This is a "maybe, maybe not' situation. In general, we don't like using autoextend in production. It allows you to be a bit lazy, not having to really monitor how objects are growing in your tablespace or how much space you have left. You can also drive your system administrator crazy, as disk space suddenly disappears on him or her.

On the other hand, if you have a system that cannot possibly be allowed to go down (a 24x7 system), then you cannot take the chance that your tablespace will run out of space – which, of course, it's bound to do at the worst possible moment. In a situation like this, autoextend is a very helpful option. For example, if you are supporting an internet site and know that a promotion is about to take place, autoextend makes sense to ensure that a sudden, huge influx of transactions will not cause a tablespace to be unable to extend.

Additionally, when you don’t know how much space you will need (an initial load of data from an unknown source for example), autoextend can help you decide how to size production (Marlene – I use your RDB example here)

Use compress=y to defragment objects and improve performance (True or False?)

This is one of those myths that sounds right but isn't so we'll say that it's false.

We've talked earlier about extents and whether they are really harmful or not. What is Oracle really doing when you set **compress** = Y when performing an export? Well, this parameter tells Oracle to look at the value of the amount of space that has been allocated for the first extent and then take the space allocated for the second extent and add the values together. Oracle continues to add the space allocated for each extent to the sum it's calculating. Notice that Oracle does not examine how much of the extent is actually being used, only how much space as been allocated. When Oracle completes the summing process, it sets the value for the initial extent in its object creation script to the sum it has calculated. Therefore, you can actually end up with a value for the initial extent that is much bigger than the available amount of contiguous space on your physical disk. Besides which, it's not compressing extents that provides the improvement in performance that you see after you've rebuilt a table!

If you see performance degradation when performing full-table scans, what's really causing it? Well, let's consider how Oracle handles inserting rows into a table. When you create a table, it starts out empty. As you insert rows to the table, Oracle uses a mechanism called the "highwater mark" to tell it where the last inserted block is. The highwater mark starts out at the beginning of a newly created table and is repositioned as data is inserted. When Oracle has to perform a full-table scan, the highwater mark tells Oracle that the end of the used blocks has been reached and the scan can complete.

Okay, so let's say that you insert 1 million rows into a table and the highwater mark is sitting way out at the end of the millionth row. Now, say that you delete 999,999 rows. Where is the highwater mark now sitting? If you said "in the same place it was before the delete," you'd be completely correct. If you then perform a full-table scan of the table that only has one actual row left in it, how many blocks will Oracle examine? Yep! That's right. Oracle will scan every block in every extent out to the highwater mark. If you have a situation where the location of the highwater mark is causing performance impact, how can you reset it?

Well, there are actually two ways to reset a highwater mark:

1. Export the table data. Drop the table. Rebuild the table. Import the table.
2. Truncate the table. (This assumes that the table is empty or that you don't care about losing the data in the table.

Notice that we did not mention setting **compress** equal to YES on our export step. You see, exporting with **compress** = Y does not reset the highwater mark or improve performance in and of itself. It's dropping and recreating the table that resets the highwater mark and helps to improve performance.

Rollback Segment Information

If you're doing an import of a large table, use a really BIG rollback segment with one really large extent (True or False?)

Well, first, you have to have two extents in a rollback segment since the first extent contains header information only and is not used for the actual rollback data. So if you size the rollback segment so that it has a really large extent, you waste the space in the first extent. We used to have a Financials Application table that was really problematic to import because it kept running out of rollback segment extents so we created and brought one huge rollback segment on line since we thought that was the definitive solution. This approach seemed to work well… for a while. Then, as the table grew even larger and we moved to later versions of the Oracle RDBMS, the approach began to fall apart and we were faced with import failures. We went back to using multiple smaller rollback segments and found that, in later versions, Oracle can make use of multiple rollback segments when doing large loads of a single table. So, by having only one extent, you may be hurting yourself and slowing down your import.

The other alternative that we had was to set **commit=y** to commit after each buffer's worth of data was loaded. We did not like this approach because, if the import failed in the middle, we would be left with an object in a questionable state.

Remember that what worked in one version of Oracle may "shoot you in the foot" in another version. As DBAs, we must be willing to try different approaches to solve problems in each new version we install.

Create rollback segments with only two extents (True or False?)

False. While Oracle places multiple transactions in each extent of a rollback segment, having only one extent for data can still be a problem. If you use all the blocks in that extent for the transactions within the rollback segment, you will have to extend the rollback segment if a transaction needs more space. This causes additional overhead as Oracle creates the new extents and performs the bookkeeping necessary to track them and what's in them.

Cary Milsap did some research and came up with a value of 20 extents as an optimal amount of initial extents to allocate for rollback segments. In earlier Financial Applications, this value seemed to really be optimal. However, with every system that you manage, you must gather the statistics and see what's really going on to decide what works best for your environment. Here are some scripts that you can use to show the rollback segment activity in your database and suggestions on actions you can take based on the values you receive.

```
Rem This script shows the percent of waits. If it's greater than 5%, add more rollback segements

select (sum(waits)/sum(gets)) * 100 Wait_Pct

from v$rollstat;

Rem This script shows the number of extents and shrinks for each rollback segment.

Rem The goal is to have extends near zero. If they are not, increase the size of the initial and
next values

Rem and rebuild the rollback segments.

select r.usn, n.name, r.extends, r.shrinks

from v$rollstat r, v$rollname n

where  r.usn=n.usn
```

Create all rollback segments in the SYSTEM tablespace (True or False?)

False. This causes fragmentation in the SYSTEM tablespace and could potentially use all the space in the SYSTEM tablespace, causing the extension of a SYS object to fail. The only things that should be in the SYSTEM tablespace are the objects owned by SYS.

About Tuning

Tune constantly (True or False?)

Yes, you can spend your entire life tuning your database and trying to tune it to the nth degree. Will you really get that much benefit from your constant tuning? Probably not! The reality is that the biggest performance gain you will get is to ensure that the application SQL is effectively tuned. If the SQL isn't tuned, you can spend every waking minute of the rest of your life trying to tune your database and possibly not see a great performance gain. Now, assuming that your SQL is really well tuned, let's look at tuning the database.

When you begin to tune a database, there are two approaches you can use. The first is to gather statistics over time and zeroing in on one metric at a time to tune. The other approach is to evaluate where your heaviest contention within the system is and try to improve or eliminate it. We suggest that you look at the Web site www.orapub.com to learn the second approach. There are scripts in almost every DBA technical book you can find that will help you gather statistics using the first approach.

Either way, though, you must first determine what your baseline database looks like. If you have known performance problems, we urge you to decide, in a measurable way, what the results are that you are trying to achieve. Once you know your goals, it is much easier to know when you've reached them so that you can stop and re-evaluate what your database and system performance is.

Also try to remember Pareto's 80/20 rule which states that a minority of causes, inputs, or efforts usually produce a majority of the results, outputs, or rewards. In other words, you want to look for the things that take the least amount of effort but achieve significant tuning results.

Always have at least three copies of the control file and two copies of the redo logs – all on different disks (True or False?)

True… Sort of. This is one of those statements that we've come to believe is true without really thinking about the circumstance under which it might no longer apply without some other parameters being added. In 1991 at the IOUW conference, Cary Millsap presented a paper on Optimal Flexible Architecture (OFA). Many of the concepts presented in this paper still apply today. One of them was that you want to have at least three copies of your control file and two copies of the redo log files on separate disks. What makes this statement false is the lack of the phrase "on different controllers."

There are two things to consider when looking at this myth. The first is with regard to Windows NT and Unix systems. You see, on a Windows NT or Unix system, you may only have one very large physical disk that has been divided into several logical disks. In this case, having more than one control file or redo log set on different disks might not buy you anything in the way of a recovery aid. If the only real, physical disk that you have fails, it isn't going to matter how many copies of what files you have on different parts of that disk. The bottom line is that you aren't going to be able to get to any of the copies.

Create all your tablespaces in one directory (True or False?)

We've been taught for years that the Optimal Flexible Architecture approach is the "only way to go" and that files must be spread across disks and across different controllers to make performance gains in I/O. But, what about the new storage technologies like EMC?

In reality, another paradigm shift may be in order for this myth. With the new storage devices now available, you may have little or no real control over where data is really placed on disks. All of the storage allocation work is done by the storage software. Therefore, where you logically place tablespaces on a storage device may have little effect on the database performance. It may turn out to be perfectly all right to place all of your files on one logical directory because the data may actually turn out to be stored in many different locations on many different disks.

Always set timed_statistics = TRUE (True or False?)

Maybe or maybe not. Let's consider what setting the parameter **timed_statistics** to TRUE in your init.ora file really does. When you enable this parameter, Oracle fills in values in some of the V$ view columns that it normally won't. For example, if **timed_statistics** is not set to TRUE, the TIM and TM columns of the V$FILESTAT view will be filled with zeros instead of real values. Also, in order to be able to process Explain plans to help evaluate and tune SQL statements, you must have **timed_statistics** turned on.

On most systems, enabling **timed_statistics** causes negligible performance degradation and gives you a great deal of helpful tuning information. However, if you are running HP-UX systems, you might want to consider only enabling this parameter on a limited basis because there have been times when Oracle has said that enabling this parameter on some of the HP-UX systems has produced a non-trivial performance degradation.

Best Practices

Don’t bother to script anything unless you will be doing it at least once a day (True or False?)

False. I don’t know about you, but I have found that those ‘quick and dirty” scripts I write, the ones I **know** I’ll never need again, are the ones I have to write over and over and over again. So I’ve learned to script everything, and keep copies. First, it allows me to eye-check what I am doing, and with luck, find any errors before I mess up my database. Second, I have a record of exactly what I did, and if I’ve made typing errors, I have a file to edit, rather than having to type it all in again (and introduce different typing errors)

Don't monitor your database. Your users will let you know about problems (True or False?)

False. Unless, of course, you like those middle of the night phone calls, with your manager screaming at you as you hear the users screaming at your manager in the background. I work better when I am not woken from a sound sleep, so if I know that I am (as an example) running out of space, I'd rather add a datafile before I need it, so that I can plan where I am putting in and when I am adding it. There is usually less impact on your database if you are being proactive rather than reactive.

Put all your application schemas into a single tablespace (True or False?)

More Tablespace Info

Set temporary tablespace to "permanent" (True or False?)

Within ORACLE as of version 7.3, you have the ability to define a tablespace as either of type "temporary" or "permanent". By default, all tablespaces are created as type **permanent**. A tablespace that is defined as type **temporary** can be used only for sort operations. Objects like tables, view, synonyms, triggers, indexes, etc. cannot be stored in a tablespace which has been defined as type **temporary**.

When you create a user in your database, you designate a default tablespace where the user's objects will be stored if he/she does not specify a tablespace name in the create statement. You also designate a temporary tablespace that will be used for the user's sort operations. Even if you do not define a tablespace as temporary, any tablespace that is identified as a user's temporary tablespace should never be used to store any objects. If you have an application in which a great many sorts are being performed, you might consider creating a separate tablespace for that application and modifying the application schema to point to that tablespace for sort operations. You will gain performance benefits from placing the tablespace on a different controller and defining the tablespace as temporary.

All operations which use sorts - like joins, index builds, ordering (ORDER BY), computations of aggregates (GROUP BY), and the ANALYZE command - will benefit by using tablespaces designated as type **temporary**. ORACLE processes its management operations differently against these tablespaces than ones that are type **permanent**. Some research that was performed showed that:

1. If a tablespace is created with the type temporary, ORACLE will not allow creation of permanent objects like tables, indexes etc. in the tablespace. ORACLE will return an error if any attempt is made to create any objects. This ensures that such tablespaces are used only for their real purpose as temporary tablespaces.
2. If you try to alter a tablespace that is of type permanent to change its type to temporary, and the tablespace already contains any permanent objects like tables, indexes, etc. then the create statement will return an error. Only when you clean up the tablespace either by dropping or moving the objects, will ORACLE allow you to change the type to permanent.

Never have more than one temporary tablespace (True or False?)

False. Some of your users may run reports that do a lot of sorting, for those users, you want temporary tablespaces with large extents, while for others, you may want small extents to be allocated.

Also, on occasion, you may need to test a large load or index. By creating a temporary tablespace just for yourself, you can later drop it when you don't need it. If you had simply enlarged the temporary tablespace that everyone else uses, you would be stuck with the extra file.

Never make your temporary tablespace extents larger than 1/2 Meg (True or False?)

False. This myth comes from more than one DBA that we've talked to that really believes all temporary tablespace extents must be the same size and must be no larger than ½ Meg. Hmm, with **autoextend** turned on for the tablespace and **maxextents** set to UNLIMITED, you might be able to get away with having small extents but, does that really make sense?

What makes better sense is to perform an evaluation of how much space the majority of your sorts are taking and setting your initial extent to that value for your temporary tablespace. If you get occasional really large sorts, set your next extent value to a much larger value and your percent increase to a high percentage. In this way, the majority of your sorts will be performed in the first extent of your temporary tablespace. If you have a really large short hit your tablespace, the extents will grow rapidly to accommodate the sort in a limited number of extents.

Of course, you must ensure that there is enough disk space available to support large extent growth. Also, if you have set the type on your tablespace to **temporary**, you must keep in mind that Oracle will not de-allocate the larger extents until the database is rebooted. If you have enough disk space to support the extents, there is no problem. If you don't, you might want to ensure that your temporary tablespace is set to type **permanent.**

Backups/Exports

If you backup your datafiles, there is no reason to do a full database export (True or False?)

False. While a backup will help you recreate your database or recover from a disk failure, it doesn't help you if you accidentally drop a table. Also, a full database export (with rows=no) is a wonderful way to document everything in your database.

If you do a full database export, there is no reason to do a files-level backup (True or False?)

False. An export is a point-in-time only data capture of your database. While you can rebuild from it, you cannot roll forward with archivelogs if you do. And an import of a full database export will take a good deal longer as a recovery mechanism than a files-level backup will.

Always leave your tablespaces in backup mode (True or False?)

False. While your datafiles will still get updated if you do this, you are spending more time and space writing to the redo logs, as Oracle writes full database blocks, not just change vectors, to the redo logs when a tablespace is in backup mode. And in versions of Oracle prior to 7.3, if you left your tablespaces in backup mode, and the database crashed, you would have to do recovery to open the database. With 7.3 and later versions, you can check the V$BACKUP view and see which tablespaces have been left in backup mode. You can then alter the tablespaces out of backup mode and open the database without recovery.

Once you create a standby database, there is nothing that will cause it to fail (True or False?)

False. Any action on the primary database that is done as "unrecoverable" will not be carried over to the standby database. In addition, if you add a datafile to a tablespace or create a new tablespace, the standby recovery will fail. You must either do a backup of the datafile or tablespace and copy it over to the standby, or manually reproduce the addition on the standby database itself before attempting to perform any recovery on the standby database. You see, if recovery is attempted and the new object does not yet exist, the recovery will fail.

Incremental exports only export the changed rows (True or False?)

False. The problem with incremental exports is that the entire table is exported , even if only one block out of millions has been changed.

About the Authors

Marlene Theriault has been an Oracle DBA for over 17 years and is currently an independent. She has been published in magazines and conference proceedings throughout the world. Ms Theriault was awarded "Best User Presentation" at EOUG in June, 1999; tied for first place with Steve Feuerstein receiving the "Outstanding Speaker" award at ECO'98; and has received the "Distinguished Speaker" award for presentations at both ECO'95 and ECO'96.

She is responsible for re-activating the DBA Special Interest Group for the Mid-Atlantic Association of Oracle Professionals (MAOP), is current Chair of the group, and publishes an "Ask the DBA" column for the MAOP newsletter.

Her books include:

Oracle8i Networking 101, Osborne McGraw-Hill (Oracle Press), September 2000

Oracle DBA 101, co-authored with Rachel Carmichael and James Viscusi, Osborne McGraw-Hill (Oracle Press), November 1999

Oracle8i DBA Handbook, co-authored with Kevin Loney, Osborne McGraw-Hill (Oracle Press), October 1999 (acting as co-author)

Oracle Security, co-authored with William Heney, O'Reilly and Associates, November 1998;

Marlene can be reached at: mtheriault@mindspring.com.

Rachel Carmichael has been an Oracle DBA for 10 years and is currently a Senior Oracle DBA for getmusic.com. She has been published in conference proceedings and presented at various international conferences. Ms. Carmichael jointly received the Distinguished Speaker award at ECO'95 with Ms. Theriault. She chairs the DBA Special Interest Group for the New York Oracle Users Group.

Her books include:

Oracle DBA 101, co-authored with Marlene Theriault and James Viscusi, Osborne McGraw-Hill (Oracle Press), November, 1999

Oracle SQL & PL/SQL Annotated Archives, co-authored with Kevin Loney, Osborne McGraw-Hill (Oracle Press), September 1998.

Rachel can be reached at rachel.carmichael@getmusic.com