Database Best Practices

DB TLC - Avoiding Complacency

Data Management Group

PROGRESS SOFTWARE

Introduction

Having been refining their database continuously for the past 20 years, Progress knows a thing or two about the advantages of not resting on your laurels. The following document makes various practical recommendations to DBAs to assist them in their efforts to provide worry-free computing for their users. The overall message we are giving through this paper is "Avoid Complacency".

Now, read on…

DB TLC

Avoid Complacency

"The days when being a database administrator was seen as a full time job are gone".

I first heard this sentiment several years ago. Needless to say, it came from the mouth of a manager negotiating for people's time at a resource meeting. The thing that stunned me when he boldly made his announcement was that he had started out as a database administrator himself, before being moved from a technical role to a managerial one.

My boss – for it was the way in which *I* spent my time that he was questioning – was admittedly "old school". He had been a techie when database administrators were constantly tending their delicate charges, like over-attentive mothers fretting over a sickly child. I understood then, and I understand now (a dozen or so years later) that his perception of my easy life was correct in his eyes; it is all relative. To him, DBAs had never had it so good.

He was right, and the trend towards ease of use continues. Nowadays, most databases are extremely reliable, fault tolerant, and self-tuning, and they have the capability to grow to previously unimaginable sizes quite comfortably.

It is not surprising, perhaps, that I am hearing similar sentiments being expressed more and more often. There is a perception issue, you see? Since databases are sturdy beasts that appear to function perfectly well without any (visible) tinkering, there appears little reason for DBAs to be involved beyond some initial planning, occasionally adding tables and fields that application developers hadn't considered during analysis, and those all-important, on-going back-ups.

Most end users are a fairly resilient lot. They tend to accept what they are given, and accept intermittent performance troughs with a resigned shrug. However, if the performance starts to have an on-going impact on the users, they will begin to grumble. So DBAs everywhere are in a difficult predicament: if they do their jobs well, their colleagues think their job is "a piece of cake"; if they don't do their jobs well, then users will soon complain about the resulting degradation in performance.

Let's make no mistake about it: degradation in performance on an unmaintained DB is inevitable. There is a scientific concept known as entropy. Entropy is generally described as a tendency towards disorder within a closed system. In laymen's terms: stuff degrades. This can be applied to almost any aspect of life. It can be observed in uranium, in a fish left out in the sun, and even in databases.

The reason is simple enough: databases are not static. They grow, its structure changes over time (remember the analysts who hadn't considered table X or field Y?), the number of users changes, the hardware degrades, etc., etc., etc., ad nauseam. Therefore, regular monitoring, careful consideration, and ad hoc maintenance is crucial in order merely to continue to give your colleagues the impression that you spend all your time reading IT magazines and surfing the Web.

Setting Up Your Database

Initial DB setup is an important time. Therefore, DBAs are generally allocated sufficient time to gather information, perform the appropriate calculations, and carefully consider options. Load testing might even be undertaken to allow you to monkey around with various settings and to measure the results. As a result, DB setup is well thought through and fit for purpose.

However, time marches relentlessly on, and entropy with it. It is common sense to revisit DB setup even though there is no pressing need – so as to avert sudden (and urgent) future needs or even to alleviate them entirely.

There are obvious times when you could revisit the DB setup, such as upgrading your DB, or recovering from a failure. You should, however, have periodic rethinks and investigations to discover whether or not the database that you (or (taxingly) your predecessor or (almost unthinkably taxingly) the predecessor of your predecessor) set up is still fit for the job.

The following guidelines are not definitive, but have proved to be the most common causes of performance problems and unrecoverable disasters for a few extremely unfortunate DBAs.

1. Consider carefully the use of startup parameters. Critical startup parameters that generally should be implemented are `-spin`, `-Bp`, `-Bt` and `-bibufs`.
2. Set `-B` as high as is possible in order to achieve a `Buffer Hit %` as far above 95% as possible (or until it starts paging). By checking the `Buffer Hit %` through consulting `PROMON` regularly, you should ensure that it never creeps below 85% as your database matures; if it does, increase the `-B` until the `Buffer Hit %` rises above 95% again.

   .Separate your indexes and your data into different areas. Index areas should be created with 1 record per block. Data areas should be created with the appropriate number of records per block. (See guideline # 11 for how to calculate the number of records per block for data areas.)
   It is also important to note that the larger the records per block the smaller can be the area size. For example, specifying 256 records per block limits the total area size to 64 gigabytes.
3. Do not have your data resident in the Schema Area. Doing so causes data and schema blocks to be interspersed.
   (N.B. This is the default for a database that is migrated from Version 8). One step in the process of migrating from Version 8 to Version 9 or Release 10 should be to spread the data and indexes appropriately across data areas that allow for more efficient storage of records. This does not have to be done all at once. It can be planned and executed over a period of time.
4. Adopt Type II storage areas. These provide better I/O performance and space allocation, and higher concurrency during allocation. It also gives 40% more throughput for database-intensive applications, and it can be implemented *ad hoc*, since you can use both Type I and Type II together.
5. Spread file extent locations across disk spindles so as to even out I/O across disks. The goal is to have all of the disk spindles performing at the same level of activity.
6. Always use fixed-length extents. Variable length extents cause unneeded performance hits when the extent must be grown. Progress has to request the blocks from the operating system and the blocks will probably not be in contiguous space on the disk.
7. Always have a variable-length extent. Every area should have a variable length extent. This is to prevent the database from crashing in the event that all of the fixed-length extents fill. The size of the area in use should be monitored and a fixed-length extent added PRIOR to growing into the variable-length extent.
8. Enable large files. This provides the ability to have variable-length extents that exceed 2 gigabytes. In the event of a database growing into the variable-length extent (See guideline # 8) the extent can then grow beyond 2 gigabytes. If the extent reaches 2 gigabytes, and large files are not enabled, the database will crash.
9. Set your database block size to be either 4K or 8K. This will allow for better performance through more efficient disk reads from the operating system.
10. Carefully consider an appropriate number of records per block to be set. Always round up to the next higher level. The different records per block values are 1, 2, 4, 8, 16, 32, 64, 128, and 256. A good formula to start with is:

    (blocksize – 64 – create limit (150)) / record size + 4
11. Avoid putting After Image and Before Image on the same drive or file system. Neither should you place either of them in the same place as data extents. After image and before image files have the most write activity for the Progress database. They are sequential writes (unlike the database, which tends to be random writes).
12. Endeavour to put Before Image onto the fastest disk: Before Image is always enabled, and it causes a significant amount of I/O. You can further seek to improve Before Image impact by running a Before Image Writer, providing a minimum of 30 Before Image buffers, and increasing the Before Image cluster and block sizes when possible.
13. Keep `Before Image writes per second` low. Check this from time to time by using the `PROMON R&D` option and studying the `Before Image Log Activity` report. To reduce the `Before Image writes per second`, you should increase the `Before Image Buffer Size`.

Disaster Recovery

The disaster recovery strategy for your DB should also be carefully considered…and periodically reconsidered. Remember that the business' expectations - in terms of data availability or acceptable recovery times from an outage - can change.

If you do not continuously ask the business to reiterate their requirements, and consider these in conjunction with new and emerging disaster recovery techniques, you could find that what you are doing no longer "cuts the mustard". If this happens after a disaster has occurred, then the people who assume your job is simple because they never see your face may end up calling for your head.

The following guidelines should act as reminders about the basics, but you may wish to consider such technologies as clustering, replication, and managed standby systems as well.

1. Where funds and scalability requirements allow, implement RAID 10 (a.k.a. RAID 1 + 0). RAID 10 is a striped array (RAID 0) whose segments are mirrored (RAID 1). RAID 10 has the same fault tolerance and overhead for this as RAID level 1, achieves high I/O rates and can frequently sustain multiple drive failures. It is ideal if you would have otherwise gone with RAID 1 but need some additional performance boost.
2. Do not use RAID 5: it increases writes to all of the disks involved in the stripe. At a minimum, there are three times as many disk writes with RAID 5 as without!
3. Have in place a well-considered database backup strategy, using any appropriate combination of the many optional approaches provided by Progress. When recovery is required, it is the public face of database administration – and a *very* public it is, too! When the moment comes, you are like a lone actor on stage. The crowd are waiting to either pelt you with rotten fruit or shower you with flowers – although they are not likely to be heard calling for an encore.
4. Print out and keep in your wallet the section in the Progress Database Administration documentation which simply states:

   "*Be sure to test the backup before you need it to restore a database.*"

   Database backups, whether full or incremental, should be verified, and this should be done as soon after the backup was taken as is operationally possible.
5. Once checked, the media needs to be archived. Label each backup volume, stating the type of backup (incremental or full, online, or offline), the date and time of the backup, the full pathname of the database, the volume number and total number of volumes of the media, and the exact command used to back up the database.
   A good rule regarding the length of archiving is to keep a minimum of 10 generations of full backups, with daily backups being retained for at least two weeks, weekly backups for at least two months, and monthly backups for a year.
   Store your backup media in a place that takes both safe distance and urgent retrieval into account. Backups should be kept away from the computer to spread the risk, but close enough for it to be delivered into your hands within a reasonable period – no point having it buried in an underground cavern in Siberia unless your users like their data a little on the stale side.

Ongoing Analysis and Maintenance

The job that most people ignore and avoid is maintenance. Think of your car. Cars are extremely complex pieces of machinery. They are frequently mind-bogglingly expensive. They carry us and our loved ones along strips of asphalt and tarmac at unnatural speeds, alongside and past hundreds of other large, heavy objects travelling just as fast. In short, they are kind of dangerous, when you think about it.

Despite this, most of us will neglect the upkeep of our cars. We may well check the tire pressure and the oil from time to time. We will refill the windscreen washer reservoir when it runs out. We probably hand it over to a mechanic every 6 months or so to have a full service. However, we do not perform daily or weekly health checks on our cars, our washing machines, or our televisions. If you do, then you are regarded as a fanatic or an enthusiast.

It is for the lack of upkeep that cars and washing machines and televisions are replaced so frequently.

You are your company's DBA. Therefore, you should be regarded as the fanatic and the enthusiast. You should perform the following tasks and consider the following advice regularly. If you do, your users could experience years of worry-free database transactions.

1. Run `dbanalysis` on a monthly basis and use the output to carefully ponder data scatter, index utilization, and fragmentation statistics.
2. Endeavour to run the background tasks known as Asynchronous Page Writers. These can be started and stopped at your whim without closing the database by using the Progress Explorer or the `proapw <db-name>` command. Asynchronous Page Writers improve database performance on shared-memory systems by performing overhead in the background. They use the resources normally associated with a user, and connect to the database as a user, but do not count as a *licensed* user.
3. Use the correct number of Asynchronous Page Writers. Two is generally the correct number. To monitor these, check for buffers being flushed at checkpoint using the legendary PROMON R&D option. If such flushes are occurring, add another Asynchronous Page Writer and recheck.
   Revisit this from time to time.
4. Appreciate that Asynchronous Page Writers ensure that a supply of empty buffers is available so that the database does not have to wait for buffers to be written to. They also reduce the number of buffers that the database engine must examine before writing a modified buffer to disk, and reduce overhead associated with checkpointing because fewer modified buffers have to be written to disk when a checkpoint occurs.
   Happily, Asynchronous Page Writers have been designed to be self-tuning.
5. Endeavour to run a Before Image Writer background task for each database. You can start and stop these at your whim without closing down the database, by using the Progress Explorer or the `probiw <db-name>` command. This background task uses the resources normally associated with a user, and connects to the database as a user, but does not count as a licensed user.
6. You should run the background task watchdog process known as PROWDOG. This monitors your database and performs some of the work for the broker process, allowing the broker to focus on more important tasks. It also monitors the broker processes for availability and consistency.
7. Endeavour to run your database with After Imaging enabled (using `rfutil` *`db-name`* `-C aimage begin`) and with the After Image Writer running. You can start and stop these at your whim without closing the database, by using the Progress Explorer or the `proaiw <db-name>` command. Backup After Image files each time you perform a backup.
8. There is no reason to leave After Image files in a full state. Mark the After Image file as empty after you have backed it up, so that the database engine can reuse it. Use `rfutil` *`db-name`* `-C aimage extent empty [` *`extent-number`* `|` *`extent-path`* `]` to do this**.**
9. If using SQL-92, run UPDATE STATISTICS weekly. This updates the statistical tables that the SQL engine uses to determine which index to use to resolve a query. These are not automatically updated.
10. If using SQL-92, start a 4GL and a SQL broker. 4GL clients perform differently from SQL clients. Having a broker process to manage each of them allows you to make appropriate broker settings for each set of clients.

By readdressing fundamental concepts and performing regular, thoughtful maintenance, your DB could hum along for years. The dilemma is that, if nothing goes wrong or degrades noticeably, you will not be thanked. You will be taken for granted. This, however, is the measure of a successful Progress DBA.

Finally, remember the greatest and most powerful guideline of them all:

1. **Murphy's Law: "If anything can go wrong, it will"**

Corporate and North American Headquarters
Progress Software Corporation, 14 Oak Park, Bedford, MA 01730 USA Tel: 781 280 4000 Fax: 781 280 4095

Europe/Middle East/Africa Headquarters
Progress Software Europe B.V. Schorpioenstraat 67 3067 GG Rotterdam, The Netherlands Tel: 31 10 286 5700 Fax: 31 10 286 5777

Latin American Headquarters
Progress Software Corporation, 2255 Glades Road, One Boca Place, Suite 300 E, Boca Raton, FL 33431 USA Tel: 561 998 2244 Fax: 561 998 1573

Asia/Pacific Headquarters
Progress Software Pty. Ltd., Level 2, 25 Ryde Road, Pymble, NSW 2073, Australia Tel: 61 2 9496 8439 Fax: 61 2 9498 7498

PROGRESS SOFTWARE

www.progress.com