

A Day in the Life of a CA-Ingres DBA

This beginner-level presentation describes the real-world operational activities of a CA-Ingres database administrator (DBA) in the form of a check-list. It describes explicitly, and in one place, all the things you need to do to keep your system running reliably, and to be able to recover from almost any conceivable disaster. As well as information that is buried somewhere in the *Database Administrator's Guide*, this presentation includes the useful---even essential---practices of the experienced DBA that are only vaguely hinted at in the manual, if they are mentioned at all.

Directives vs. Good Judgement

Most of the advice given here is in the form of directives. This was done for the benefit of the completely green Ingres DBA to save having to go into excessive detail about the reasons for certain tasks that the tyro is not equipped to understand anyway. Those of you with a little bit of experience under your belt may feel that some of the advice given in this presentation is excessively cautious, or irrelevant in some situations. If you are sufficiently knowledgeable to make an *informed* decision not to perform some activity recommended here, then you are not the audience the recommendation was aimed at; good judgement must always be allowed to overrule policy. This presentation merely suggests some good policies that may not be obvious from reading the manuals.

(However, if you feel disinclined to follow the advice that is contained here just because it seems inconvenient or it doesn't suit you, but you can't provide a reasoned argument why it is unnecessary, you should by all means take this opportunity to find out more about why it is recommended. It may save you a nasty shock later.)

Where Do I Start? A Hierarchy of Priorities for the New CA-Ingres DBA

The first thing the new DBA has to decide is where to start. Very often you will discover that you have competing or even antagonistic demands on you. To help you decide what to do when faced with competing demands or limited resources, or even limited expertise, the following hierarchy of priorities (in descending order of importance) is suggested.

- 1 Safeguard the data
- 2 Secure the data
- 3 Ensure adequate performance
- 4 Model reality
- 5 Document procedures
- 6 Fine-tune for ultimate performance

Interestingly, the amount of sophistication required for each task increases as its importance decreases (more or less). This is good news for the beginner. The most important tasks can often be performed completely effectively just by rote, giving you time to develop your expertise before attempting more challenging tasks.

Possibly this hierarchy requires some explanation and needs to be defended.

Safeguard the Data

The new DBA's first priority must be to safeguard the data. It is almost inconceivable that any organization would take on the considerable expense and trouble of acquiring, installing, and programming a database system if the data stored in it were unimportant. Your first priority, if you can do nothing else, is to ensure that no data is ever be lost or damaged. Storing data is the *raison d'être* of the database. If you cannot guarantee that no data will ever be lost or corrupted then none of the other tasks are worth even bothering with. What would be the point of tuning the server for faster access to rubbish--rubbish that may have vanished anyway?

The new DBA should therefore concentrate on learning about backups and recovery, journalling, and data integrity before doing anything else. It would probably also be appropriate to review your data centre's system backup procedures as part of this job. Make sure that you understand what your operations people are doing and that it meets your expectations. Make sure that they keep at least some of the back up tapes off-site. Satisfy yourself that the computers, storage devices, and off-line media (tapes probably) are reasonably secure from fire or flood. (For example one of my clients had their entire data-centre housed in a room adjacent to a welding shop and did not keep their tapes off-site. Another had a 14" water main that ran through the basement machine room--it burst and filled the room chest-deep with water and asbestos fibres, and the UPS batteries leaked.)

As well as doing backups and checkpoints, make sure that you know how to recover using them. Practice the procedure as often as feasible, but certainly no less than twice a year. Ask to rent time at a hot-site twice a year for this purpose if you do not have a development system to practice on. If you don't know *for sure* that your checkpoints work and you don't know what to do with them anyway, that is as good as not doing them at all.

As you can see, most of the activity required to ensure the safety of the database is continual, requiring active daily intervention by the DBA (if not in person, then at least through processes defined and scheduled by the DBA). Other activities can be infrequent, but will probably be manual and quite intensive. All of them are listed in the check-list provided later.

Secure the Data

After ensuring that the data is safe from disasters of every imaginable sort, the next most

important task for the new DBA will be to ensure that no one can alter the data without proper authority, and to ensure that no one can see the data without proper authority. Which of these two is most important will probably vary depending on the organization and probably even on a table by table basis. For example, in a medical application legal obligations may make it slightly more important that identifying patient information be hidden, and slightly less important that appointment times should be protected from being altered. In a payroll system it may be more important that pay scales cannot be updated, and slightly less important that they be concealed.

Note that we are talking only of *relative* importance. Obviously we should aim to impose every sensible security constraint consistent with usability. The point here is to establish a hierarchy of priorities, to answer the question: you can't do it all at once, where do you start? This decision will be determined by estimates of the harm resulting from unauthorized exploitation of a privilege which should not have been granted.

Having established security rules, it then becomes a periodic maintenance task to review security. Requirements change with time; roles change with time, and table definitions change with time. Our check-list therefore includes periodic security audits.

Fine -Tune for Ultimate Performance

It is not clear that attempting to fine-tune for ultimate performance should be subordinate to maintaining documentation. For example it would be very hard to explain to your boss why you are writing about backup procedures while 100 users are experiencing response times in excess of a minute. On the other hand, it would be equally hard to explain why no one could recover the database while you were on vacation when you had six months of adequate and reliable system performance and nothing else to keep you from documenting your procedures. There is a boundary here, but it is not completely clear just where it lies.

To help to resolve it, fall back on the priorities that are already established beyond dispute. Response times that are so bad that the users are unable to carry out their job functions are really very nearly equivalent in practical impact to losing the data or not having permission to access it. This is not really a matter of *enhancing* performance, this is a matter of making the system minimally usable. Resolving this problem is therefore clearly a very high priority that overrides documentation.

On the other hand, "twiddling the knobs" with the expectation of shaving a mere 5 seconds off a 3 hour batch process is no real help to anyone. Every DBA enjoys doing it, of course, but it seldom yields any discernible benefit to any given user. Just because you can measure it doesn't mean they will notice it.

Server tuning is a notorious distraction of this sort. *No Ingres system has ever been transformed from a turkey to a lark by server tuning.* Fiddling with the locking and logging systems will allow you to wring the last drop of performance out of an otherwise well-designed system, but it will not solve an acute performance problem. Your company will probably never recover the

cost of the time you spend doing such fine tuning. If server tuning needs doing at all, it can wait. This is clearly an activity that should be done during periods when the DBA would otherwise be completely idle, if at all, and writing documentation certainly takes priority over it.

There are no analytical or diagnostic tasks related to tuning in the check-list. However the check-list does include periodic grooming of table structures and gathering of fresh statistics, to ensure that CA-Ingres has the best chance of operating well.

The Last Word on the Hierarchy of Priorities

Naturally you will develop your own priorities as your experience grows, but until you do, the hierarchy listed here is very conservative, and will help you to make decisions that minimize the risk of expensive or disastrous errors due to inexperience. Following the check-list will practically guarantee that you will never go too far wrong. But like the hierarchy of priorities from which it was developed, the check-list is very conservative too. It may be quite tempting sometimes to be a little less than punctillious about following it. That would be a mistake. To illustrate the virtues of a cautious and conservative approach consider the following true story about an experienced and skillful DBA who still got into trouble because he made the mistake of relying on everything working like he "knew" it was supposed to.

A Cautionary Tale

This story concerns the advice given in the check-list that you should always do a system back-up before doing a rollforward. You may feel that doing a system backup before attempting a rollforward is excessively conservative, after all, you have a valid checkpoint. But consider the following questions:

- Are you certain you know what you are doing well enough that you can guarantee that you will make no errors that will make the situation worse? (Are you certain the manual tells you everything you need to know? Are all the known possible modes of failure documented?)
- Are you certain that some of the least-frequently used components of the CAIngres system (ie the rollforward) have been exercised sufficiently by users in the field to uncover every possible mode of failure?
- Are you certain that you will not experience a power-failure or media-failure at a crucial moment?
- Is it good database management practice to do *anything* to the database that cannot be reversed if necessary?

Of course the answer to all these questions is "no". Now consider this question: what will you tell your boss, or your boss's boss if (when) your attempted rollforward fails and leaves a worse mess than you started with?

Never, ever, under any circumstances attempt any database recovery without first doing a system backup. With a valid system backup in hand, the worst that can happen when a problem occurs during a recovery is that you waste a bit more time. The worst that can happen if you don't have a valid back up is that you can lose a lot of data that your company spent a lot of money to capture. For some companies any loss could mean real financial penalties measured in hundreds of thousands of dollars.

Unfortunately, during a crisis involving the database, the new DBA will feel great pressure to resolve the problem quickly. This pressure may exist only in the DBA's imagination, or it may be real, and while there is no justification for dawdling and wasting time, nor is there any justification for rushing and taking foolish risks either. Even with management breathing down your neck demanding instant results, take the time to do the job properly, by starting with a system backup. The following true story, taken verbatim from an Internet posting on **comp.databases.ingres** may help you to maintain your resolve to do it right, even in the face of importunate management demands to cut corners. (This is reprinted here with its author's permission.)

I found this out the hard way. Nedless to say, I'm very choked, but maybe this will save you some hairs...

Scenario: UNIX system with 6.4/04 Ingres

Disk1: UNIX, page space

Disk2: II_SYSTEM

Disk3: II_SYSTEM/ingres/jnl

II_SYSTEM/ingres/ckp

II_SYSTEM/ingres/dmp

Every night, the system backs up all the above, then takes a checkpoint.

One day (late morning), Disk2 crashes. No problem, you replace the disk, then restore your backup of II_SYSTEM, being carefull not to overwrite the jnl|ckp|dmp directories. Everyone is concerned about lost work, but you tell them its no problem, Ingres has this terrific journalling...

So you do the above, then issue the rollforwarddb command to recover work from the checkpoint & journal files, expecting it to work...

WRONG WRONG WRONG WRONG WRONG!

What happens is that Ingres looks at the aaaaaaaa.cnf file in the data directory (not the dmp copy), sees that its different from the one in the dmp directory, aborts the rollforward, and without warning, mind

you OVERWRITES the file in the dmp directory!

STUPID STUPID STUPID STUPID STUPID!

Now, the checkpoint & journal you have on Disk3 are useless! You might as well delete them, because there is NO WAY you can use them. The aaaaaaaaa.cnf file in the data directory is the one the system uses, and in this case it was restored from the backup. The checkpoint and journal files can only be accessed if you have the CURRENT copy of the aaaaaaaaa.cnf file, which existed on the dmp directory, but was overwritten by the abortive rollforward.

To make matters worse, the aaaaaaaaa.cnf file which is tar"ed into the latest checkpoint (and can be extracted) is useless, since it indicates that the checkpoint it is a part of is INVALID of course, since the checkpoint it is a part of has not yet completed when it is "tar"ed...

What I learned:

- 1) Copy the aaaaaaaaa.cnf from the dmp directory to the data directory before doing rollforward Ingres must be down while this is done
- 2) There is not Ingres tool to correct the aaaaaaaaa.cnf file if it is in error; Ingres should be taken to task for this.
- 3) Make backup cpoies of your journal & aaaaaaaaa.cnf files before trying a rollforward; if the roll fails it will trash these files and anything else that's handy.
- 4) The documentation makes no mention of any of this, and probably a lot of other **very** important things as well.

Thank you. I feel much better. Can I have my bullets back now, please?

This particular DBA could have spared himself a lot of grief, and his client a lot of expense, if he had taken the time to do a system backup before attempting his recovery. If he had done so, he would have been able to get back to the original problem state with nothing worse than a bit more downtime. As it is, he ended up with more downtime *and* lost data.

A Check-List

Daily Activities; AM

=====

[] 1. Review errlog.log for previous day

Either use snifflog and grep, or rely on AutoDBA. Both of these tools will filter out uninteresting diagnostic messages that can safely be ignored for the shortterm. Respond to reported errors according to their (potential) severity, as needed.

To run snifflog for October 5th, type:

```
snifflog | grep 'Oct 5' | more
```

Note the TWO spaces in 'Oct 5', but there would be only 1 space in 'Oct 10'.

[] 2. Review checkpoint log

Make sure the checkpoints were completed as expected.

[] 3. Run infodb for each database.

Reconfirm checkpoint completed properly and that journalling is occurring.

[] 4. Review operating system dump log

Make sure the journal and dump files were backed up after the checkpoint was written. (Also make sure \$II_SYSTEM/ingres/files/symbol.tbl was backed up).

[] 5. Make sure dmfacp (the archiver) is running

dmfacp can stop for a number of reasons. Ensure that it is still running. (In the event of a log file failure the database will be corrupted and the database will only be recoverable up to the last time the archiver ran.)

[] 6. Make sure there is sufficient disc space

Use bdf or a similar tool to ensure there is sufficient disc space to modify the largest table in each location (requires free space equal to 2 times the size of the largest table), and for the database to grow (unless you are counting on AutoDBA to report disc space shortfalls).

Weekly Activities

=====

[] 1. Print off (in HARDCOPY) the contents of the iifile_info table.

Retain a hardcopy of the iifile_info table so that the name of the file underlying a table can be determined, to allow recovery from the system backup tape if that becomes necessary as a last

resort. Repeat for all databases INCLUDING iidbdb. (AutoDBA logs this same information in \$IQ_LOGDIR/*.map, so it is sufficient to just print those reports once a week.)

[] 2. Run verifydb mrun sdba opurge for all databases (including iidbdb).

[] 3. Run sysmod for all databases (including iidbdb). (The AutoDBA scripts will do this for you. The scripts are in \$IQ_LOGDIR/*.com)

[] 4. Modify tables in all databases.

Modify tables that are prone to developing overflow and "holes" (ie active ISAM and HASH tables, and HEAPs and BTREES with lots of deletions.) (The AutoDBA scripts will identify and correct these tables. The scripts are in \$IQ_LOGDIR/*.com)

[] 5. Checkpoint iidbdb, offline.

[] 6. Run unloaddb for all databases.

Run unloaddb and save the scripts, but DO NOT execute them. These are only for reference in an emergency, lastditch, salvage operation.

[] 7. Shut down and restart the Ingres servers.

Restart the servers to reinitialize all the caches and memory pools that are prone to fragmentation.

[] 8. Rotate offsite checkpoint tape set (for all databases).

Keep a set of checkpoint tapes offsite in secure storage as a hedge against total disaster (fire, flood, etc). These need not be the most recent checkpoints, since they are never likely to be needed and it would be inconvenient to have the most recent checkpoint not easily available if a rollforward were required.

Monthly Activities

=====

[] 1. Run optimizedb on all databases (including iidbdb). (The AutoDBA scripts will identify and correct these tables. The scripts are in \$IQ_LOGDIR/*.com)

[] 2. Truncate errlog.log (possibly back it up first)

Annual Activities

=====

[] 1. Practice doing a rollforwarddb

(unless you've done one for real in the last year).

Further Reading, and Other Sources of Assistance

Acknowledgements

This presentation draws heavily on the advice, experience, and writing of others. My main contribution to it has been to compile everything in one document, with only a few additions of my own (apart from my judgement of what to include and what to leave out). I particularly would like to thank the following people for their contributions and comments: Jack Russo, Maarten Veerman, Mike Leo, Alan Fleming, etc. etc.