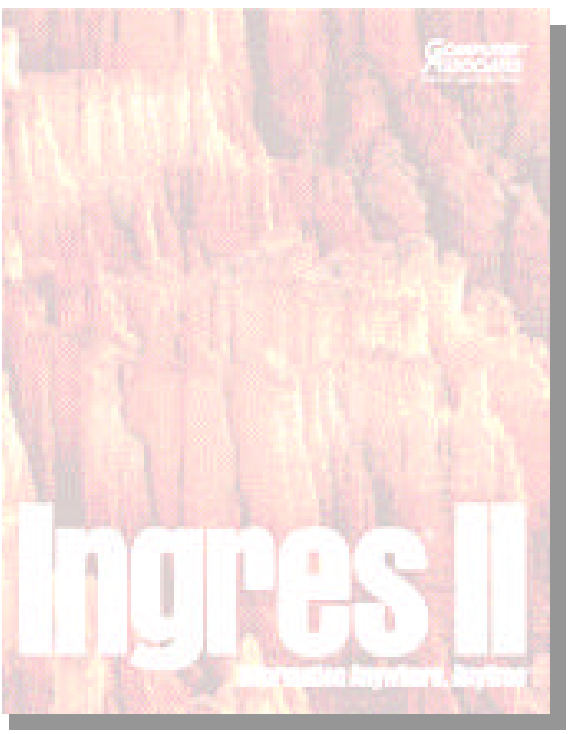
Planning and Performing Your Upgrade

A Practical Guide (Cookbook) For Ingres Users



Ingres® II

Table of Contents

Disclaimer

The Ingres II Migration Guide is intended to offer procedures that provide guidance to Ingres 6.4 users upgrading to Ingres II. Ingres II is the fourth release of Ingres since version Ingres 6.4, and many new features and capabilities are included. The procedures in this Migration Guide have been tested and are as comprehensive as is possible. However, since no Ingres installation and environment is ever identical, no migration guide can be 100 percent complete. Keep in mind that the information in this guide is not a replacement for thought, and use caution and common sense when performing upgrades at your site. The number one rule you should always consider is to make sure you understand what you are trying to accomplish with each step, and make sure you have a plan in case something should go wrong. Carelessness can result in loss or corruption of data, but with careful planning and execution, you can maximize your chances for an easy migration that will increase the performance and capabilities of your Ingres-based systems.

Credits

Special thanks to Karl Schendel and the Ingres Development and Support Teams for their hard work and dedication to Ingres and this Migration Guide. This guide has been reviewed and approved by Computer Associates International Inc.

Introduction to Ingres II

Ingres II is the only complete solution for n-tier relational application development and information management. Integrated with an industrial-strength RDBMS, Ingres II provides a full suite of options that offer enterprise access to existing data, replication, internet commerce capabilities, and a unique application development environment - OpenROAD. Ingres II offers unique solutions across Microsoft Windows NT, OpenVMS, and a variety of UNIX platforms.

Planning Your Upgrade

The most important thing you can do is thoroughly plan for your upgrade before you start. This manual, together with the online documentation set included on your Ingres II Enterprise Edition CDs will assist you in planning and executing a successful upgrade. With detailed planning, potential problems will become evident, allowing you to plan for them ahead of time, and be proactive in their prevention. Problem prevention is the key to any easy hardware or software upgrade.

Testing your plan, preferable with a copy of real data, will focus your attention on any areas that might cause problems during the live upgrade on your production systems. Implementation should not begin until all preliminary testing is complete. These procedures are basically common sense, but are often overlooked.

Create a checklist, items as simple as: How long to prepare and complete an up-to-date backup; and How to ensure that all data is complete (no bad tapes); are issues that need to be dealt with.

The best strategy for doing an upgrade is to implement any compatibility fixes in your Ingres 6.4 environment first. Once your databases and applications are Ingres II ready, you test them in an Ingres II installation, practice the upgrade, and then do the live upgrade. Use of any of the new Ingres II features are deferred until after the live upgrade is successful. This way, you minimize the number of variables at each step, and maximize your chances for immediate success.

Hardware Issues - Planning, Testing and Implementation

TIP
Backup all data before you begin.

In order to do a safe and orderly upgrade, you will need at least three Ingres installations (four preferred); your live production; your Ingres 6.4 development installation; an Ingres 6.4 installation for testing the upgrade; and an Ingres II development installation for preparing and testing your applications.

You need someplace safe to put these installations. Try to keep them off your production machine, if at all possible. If you have no available computer for preparing and testing your upgrade, you might consider temporarily getting one. It is possible to do everything on one machine, but it is more difficult and riskier.

The minimal hardware setup recommended is three machines: development, test, and production. Initially all three should have Ingres 6.4 installations. Install a separate Ingres II installation on development, and get everything working there. Then you upgrade the test machine (perhaps more than once!) for practice, and to make sure it all works. Then you upgrade the live machine.

PRACTICE TIP
Expect to have Ingres 6.4 development cease while you practice the Ingres 6.4 to Ingres II upgrade in that installation.

Another alternative involves two machines, development and production. Install a separate Ingres II installation on development, and get everything working there. Then upgrade the Ingres 6.4 installation on that machine (perhaps more than once!) for ***practice***, and to make sure it all works. Then you upgrade the live machine. (Although it is possible to do everything on one machine, using three different Ingres installations. This is not recommended. It is too easy to do something in the wrong installation.

Of course, you can also use four machines, separating the Ingres 6.4 and Ingres II development machines. This might actually be too much of a good thing, as there tends to be a lot of back-and-forth between the Ingres 6.4 and Ingres II installations, especially in the late stages of testing and compatibility fixing.

A minor hardware-related point: there is no remote installation procedure for Ingres II. If your machines do not have local media support (CD-ROM or tape), you will have to arrange it; or be prepared to copy around the distribution tar file from a central point.

Application Issues

This may sound silly, but do you know where your applications are?

It is difficult to prepare or test applications if you do not know that they exist. It is very easy to concentrate on the big, mission critical applications and miss a tiny little database -- used by the president's executive secretary.

Take an application and database inventory before you start, and make sure you know how to rebuild every application. If you find an application that you can not rebuild for some reason, be sure to test it under Ingres II as soon as possible. Sometimes it is possible to run an Ingres 6.4 application image against an Ingres II database, but only if that application has no upward compatibility issues. You may find yourself recreating that application from scratch, or doing without it.

If your site is doing active application development, you have another worry, which is how to coordinate new development with Ingres II compatibility fixing. Generally it is best to try to synchronize the test and live upgrades with an appropriate time in your application enhancement cycle. For example, one product-oriented site addressed this issue by synchronizing Ingres II compatibility with a code release; then, development was converted to Ingres II, while a Ingres 6.4 "bugfix" installation was kept around on a different machine for maintenance purposes.

Type of Upgrade

You have two options for doing your live upgrade: the use of the upgradedb utility, or an unload-reload. You can mix the two, upgrading some databases while reloading others. In fact, unless you do a cold Ingres re-install, you will upgradedb the iidbdb even if you decide to unload-reload the user databases.

The upgradedb utility upgrades an Ingres 6.4 database in place, quickly, and with no additional disk space required. Preparing for a safe and reliable upgradedb takes some time, though. Specific versions of upgradedb have some specific areas that must be allowed for.

A database unload and reload ensures that you have a clean start with a fresh database, although you do need disk space to do the unload and reload. Depending on what kind of tables you have, you might need three to five times the space that your Ingres 6.4 database took up. For instance, compressed tables with wide CHAR or VARCHAR columns can "blow up" substantially when unloaded. Also, the reload process takes longer than upgradedb and delays the availability of your installation after the upgrade.

A database that has been running for years, perhaps living through a number of system crashes and hardware failures, could have accumulated damage that might confuse upgradedb. For example, a database that is used by a small department or group of people, may not be maintained as well as a production database. To continue this example, such a database may have junk tables owned by a no-longer-existing user, with missing table data files. The upgrade procedure given here is designed to detect and cure as much of this sort of damage as possible, but it cannot correct for missing data files! If you were thinking of reloading your database anyway, the Ingres II upgrade might be a good time to do it.

Various upgrade scenario comparisons have shown that the unload/reload process is slower (due largely to additional disk activity). If you suspect data problems for any reason, you might want to perform the unload/reload. Otherwise, the upgradedb utility is the better choice.

Getting Started

Assuming that you have a separate development machine with Ingres 6.4 loaded, your first step is to install Ingres II in a separate installation on that machine. If you are machine-rich, you could even install Ingres II by itself on yet another computer, but we will assume that your development computer will support both an Ingres 6.4 and an Ingres II installation.

Here's an outline of how to install Ingres II on the development machine.

Create a new ingres directory someplace with enough disk space.

Suppose /ing20/ingres is such a directory.

```
mkdir /ing20/ingres
chmod 755 /ing20/ingres
```

Create a couple of scripts called "set64" and "set20" to set your environment to Ingres 6.4 and Ingres II respectively. Here are example scripts for the C-Shell. You may have to adjust them slightly for your installation; for instance, your PATH may need to be slightly different, and LD_LIBRARY_PATH may be named LIBPATH or SHLIB_PATH on some platforms.

```
     set64:
     setenv II_SYSTEM /ing64/ingres
     set path=(. /usr/local/bin /bin /usr/ucb
/usr/sbin
     /usr/openwin/bin $II_SYSTEM/ingres/bin
$II_SYSTEM/ingres/utility
     /usr/ccs/bin)
     set inst=`ingprenv1 II_INSTALLATION`
     setenv LD_LIBRARY_PATH /usr/lib:/usr/openwin/lib
     set prompt=`whoami`.`uname -n`"[$inst]% "
     echo "Switching to Ingres 6.4 [$inst]
installation"
```

```
     set20:
     setenv II_SYSTEM /ing20/ingres
     set path=(. /usr/local/bin /bin /usr/ucb
/usr/sbin
     /usr/openwin/bin $II_SYSTEM/ingres/bin
$II_SYSTEM/ingres/utility
     /usr/ccs/bin)
     set inst=`ingprenv II_INSTALLATION`
     setenv LD_LIBRARY_PATH
     /usr/lib:/usr/openwin/lib:$II_SYSTEM/ingres/lib
     set prompt=`whoami`.`uname -n`"[$inst]% "
     echo "Switching to 2.0 [$inst] installation"
     Define aliases (C-shell) or shell functions
(Bourne/ksh) to
     source the set64 and set20 scripts.
```

For instance:

```
     alias set64 source ~ingres/set64
     alias set20 source ~ingres/set20
```

"set20" to the Ingres II environment, and cd to $II_SYSTEM/ingres. Follow the Ingres II installation instructions to install Ingres II. Don't use the same data, checkpoint, journal, dump, or log directories as your Ingres 6.4 installation (although you can use the same disks). For instance, if the Ingres 6.4 installation uses a data area called /bigdisk, you might create a directory /bigdisk/ing20 for the Ingres II installation's data area.

If you are new to Ingres II, now might be a good time to play with a few of the new Ingres II management functions, such as cbf. You will also notice that Ingres II looks different in a "ps" listing. It does not use I/O slaves (unless you are on a platform that does not support OS threads or asynchronous I/O). The dmfrcp process is replaced by another iidbms process, the recovery server.

Initial Application Preparation

There are some applications created under Ingres 6.4 that will run unchanged under Ingres II, but you should not count on this. There are a number of changes in Ingres II that may require you to change your

applications. In general, none of the required changes are particularly difficult.

The first couple of issues should be checked for right away, before attempting to move your applications and databases to the Ingres II development installation.

New Reserved Words

Ingres II reserves a number of new keywords, mostly for support of the SQL additions implemented by Ingres II. If you used names like "level", "key", or "comment" as column names, you will need to change them. *See the Ingres II SQL Reference Manual, Appendix A,* and the *Reserved Words* list included with this document for a complete list of Ingres II reserved words.

Checking for and fixing reserved word conflicts should be the first thing you do, since you can not move your databases to the development Ingres II installations until word conflicts are edited. Remember to check for *Reserved Word* conflicts in dynamically created tables and views in your application code.

Report Writer syntax change

In order to support new syntax, the Report Writer now requires a space after all dot-commands. Thus, syntax like ".NL3" must be changed to ".NL 3". The following sed command can be used to fix most such occurrences automatically:

```
sed -e 's/\([   ]\.[a-zA-Z][a-zA-Z]*\)\([0-9]\)/\1
\2/' foo.rw | \
sed -e 's/^\(\.[a-zA-Z][a-zA-Z]*\)\([0-9]\)/\1 \2/'
>newfoo.rw
```

(That's a <space><tab> inside the [ ] in the first line.)

You should diff the old and new files (foo.rw and newfoo.rw) to make sure that nothing unexpected happened, such as an unwanted "fix" to a literal string.

Another way to alter Report Writer files is to sreport them into a database, then copyrep them back out. This might be the preferred

technique if you normally sreport your Report Writer scripts into a database regularly.

Loading The Ingres II Development Installation

Once you have gotten this far, you can try copying your database and applications from your normal development Ingres 6.4 installation into your development Ingres II installation. (This is where you will do your initial application checkouts to make sure everything still works under Ingres II). At this point, you need not carry along much data, since you probably have more work to do before your applications really work. It will be useful to carry along a small subset of test data. If you are coming from an Ingres 6.4/04 or earlier installation, you may need to deal with this issue as you move your database/application over.

Trailing quote missing from copyapp/copyform output

Ingres 6.4/04 and earlier versions of Ingres would sometimes omit the trailing quote from format strings when copying out a form. This would occur with both copyform and copyapp out. There is not much you can do other than manually fix the output files before copying the forms back into Ingres II. This does not occur in Ingres 6.4/05 and later versions.

Moving Databases

Moving the Ingres 6.4 development database from Ingres 6.4 to Ingres II, if you do not already have procedures for moving databases around:

- "set64" and cd to a directory with enough space to hold the data. Remember to allow for the front-end catalogs.
- Run unloaddb against your Ingres 6.4 database. Run unload.ing to dump out the front-end catalogs and your data.
- Edit the cp_ingre.in file and remove the line:
  `\include /ing64/ingres/files/iiud64.scr`
  (Your directory path will probably be different.)
- "set20" to the Ingres II installation. Create the database there, but without any front-end catalogs:
  createdb databasename -f nofeclients
- Edit the reload.ing script if the Ingres II database name is not the same as the Ingres 6.4 database name. Then, run reload.ing.

reload.ing TIP

TEE THE OUTPUT TO A LOG FILE IN CASE ANY ERRORS OCCUR; YOU MAY RUN INTO RESERVED WORD PROBLEMS.
`RELOAD.ING |& TEE /SOMEPLACE/RELOAD.LOG`

At this point the front-end catalogs in the Ingres II database are in the Ingres 6.4 format. To get them into Ingres II form, run upgradefe:

```
upgradefe databasename INGRES
```

The above assumes that you intend to copy the data in the Ingres 6.4 development database to Ingres II as well as the catalogs. If you do not want the data, you can edit the scripts that unloaddb creates so as to not do the COPY INTO of some or all of the tables.

Be sure to review the output of the reload into the Ingres II database. If you have any reserved word conflicts with the Ingres II Reserved Words, they will show up in the reload. Fix the Reserved Words on the Ingres 6.4 side, then try again.

If you have Star databases, remember to do a regular unloaddb on the CDB, and an unloaddb /star on the DDB. (The CDB is the coordinator database, the one that usually starts with "ii". The DDB is the distributed database, the one you usually access by saying ddbname/star.) The unload of the DDB will unload all registrations and distributed view definitions; it does not generate any table data. The unload of the CDB will unload any locally stored tables that do not exist in other local databases.

Additional Application Preparation

After successfully creating databases and applications in the development Ingres II installation, continue checking for these application issues:

UPDATE FROM semantics change

The test if ambiguous replace in Ingres up through Ingres 6.4/05 allowed the update as long as each target row was being updated with an unambiguous value. Ingres II (and Ingres 6.4/06 and later) tests for multiple FROM rows instead, and generates an ambiguous replace even if all the FROM rows generate the same replacement value. For instance, consider the (nonsense) query:

```
UPDATE foo FROM iitables
SET thing = 1;
```

Ingres 6.4/05 and earlier would allow this, even though there was no WHERE qualification joining the tables, since the replacement value was non-ambiguous. Ingres II will produce an Ambiguous Replace error.

Probably the best way to handle this, is to review all applications for potentially ambiguous updates and change them to use EXISTS or IN, instead of a join. If this is not feasible, the old UPDATE FROM handling can be restored by using CBF to set the DBMS parameter "ambig_replace_64compat" to ON.

Decimal constant semantics change

With the introduction of the DECIMAL data type, fixed point literals such as 1.0 are now considered DECIMAL, rather than FLOAT. Normally, this does not matter, as Ingres does appropriate type conversions. In one instance it is important, though, and that is when you are doing a CREATE TABLE .. AS SELECT with a constant in the SELECT result list. For example:

```
CREATE TABLE foo AS
SELECT thing1, thing2, num_thing = 1.0
FROM bar;
```

In Ingres 6.4, the num_thing column is created as FLOAT8. In Ingres II, though, a DECIMAL(2,1) column is created, which will easily overflow.

Again, the best way to handle this, is to examine uses of fixed-point constants in your applications and change them to floating point constants (1.0e0, for instance), or add an explicit FLOAT8 type conversion. A less thorough but much faster alternative is to set the environment variable II_NUMERIC_LITERAL to FLOAT.

```
setenv II_NUMERIC_LITERAL FLOAT
```

This tells Ingres II to interpret fixed-point constants as floats, not as decimals. If you decide to depend on II_NUMERIC_LITERAL, be aware that you will have to arrange for EVERY user of the application(s) to set II_NUMERIC_LITERAL in their environment.

Greater sensitivity to BYREF errors

Ingres 6.4 4GL programs were relatively insensitive to length and type errors when returning BYREF values to a calling program, particularly when a 3GL routine was called. Ingres II is more sensitive to returning values that are too long, or are of the wrong type; in some cases errors of this nature can result in program aborts and coredumps. The only cure is to correct the called routines so that they return values of the correct type, and don't try to return longer values than the calling program expects.

Journaling ON by default

In Ingres 6.4, even if a database was journaled, you had to explicitly say WITH JOURNALING to get a newly created table to be journaled. In Ingres II, journaling is on by default. This means that if you application creates (and drops) temporary tables as it runs, those tables will be journaled, wasting system resources and possibly making the Ingres II installation run more slowly than expected.

It is possible to turn the default for journaling to OFF by changing a CBF parameter (default_journaling). This might be a good solution for carefully controlled environments. Another option is to issue a SET NOJOURNALING statement at the beginning of applications that create temp tables, or to make sure that temp tables are created WITH NOJOURNALING.

Greater sensitivity to arithmetic errors

Ingres 6.4 would ignore a number of arithmetic error conditions - such as floating point overflow or divide-by-zero. Ingres II is more careful to report arithmetic errors properly on all platforms. If your application starts generating arithmetic exceptions when tested with Ingres II, it is an indication that the application code had problems previously, and needs to be corrected.

Free-space management pages

Ingres II adds a couple new non-data pages to all tables, including heap tables. These pages support improved free-space management by Ingres II. Normally this is not an issue. For most users this will not be an issue, however if you application works closely with TIDs, you may need to change your application to expect gaps in the TID page number sequence. Working with TIDs in this fashion is not recommended.

4GL TABLE_KEY type conversions

Conversion of 4GL VARCHAR variables to the TABLE_KEY type gives length errors. Avoid this by converting to char first:

```
TABLE_KEY(CHAR(varcharVariable))
```

User Defined Datatype changes

If you are using the Object Management Extension and declaring your own User Defined datatypes in the server, there were some changes in calling sequences. Read the Ingres II Object Management manual for details.

Report Writer runtime parameter errors

This is not really an application issue, but it looks like one. If you attempt to pass parameter strings containing quotes to the Report Writer, you may experience mysterious looking runtime parameter errors. This can be caused by a quote change to the UNIX command parameter passing control file, utexe.def. If this occurs it may be simplest to restore the original utexe.def file quoting, as follows:

```
Edit $II_SYSTEM/ingres/files/utexe.def
Do a search for the string '(%S)'
Change it to: param  '(%S)'
```

Save the file and see if the error goes away.

Remember that most of the changes required to prepare for Ingres II are backward compatible to Ingres 6.4. It is prudent and desirable to make the application changes in the Ingres 6.4 installation, and bring them forward to the Ingres II installation for testing. This way you do not have to freeze development while preparing the applications, and you have maximum flexibility.

It is very tempting to slip in some Ingres II-only code as you are reviewing your applications or addressing compatibility issues. Resist the temptation! While an outer join or a session temp table in a crucial place may do wonders for performance, there is plenty of time to add speed *after* the upgrade.

System Administration Preparation

A number of Ingres II upgrade issues involve system or Ingres administration. You will need to coordinate these changes with your system administrator.

Shared library search path

Ingres II uses shared libraries on many UNIX platforms. Since there is no standard default installation directory for Ingres II, you will have to tell applications and tools where Ingres II was installed on YOUR machine so that the shared libraries can be found. Generally, this is done in one of two ways: LD_LIBRARY_PATH (or equivalent), or linking to /usr/lib.

You can choose to define the LD_LIBRARY_PATH environment variable to include the Ingres library directory, $II_SYSTEM/ingres/lib. Some platforms use the variable name SHLIB_PATH or LIBPATH instead of LD_LIBRARY_PATH. All users who access any Ingres programs or applications will have to define LD_LIBRARY_PATH. Failure to have LD_LIBRARY_PATH set will result in an error message that looks something like this:

```
ld.so.1: /ing20/20/ingres/bin/tm: fatal:
libframe.1.so: open failed: No such file or directory
```

It does not hurt to include $II_SYSTEM/ingres/lib in LD_LIBRARY_PATH for Ingres 6.4, so your system administrator can get started on this change as soon as possible.

A more secure and generally easier way of accessing the shared libraries is to link them into /usr/lib. For example:

```
ln -s /ing20/ingres/lib/libframe.1.so /usr/lib
```

This does not require any application wrappers or user environment changes. The downside of this approach is that your system administrator has to individually link each Ingres library. You also have to review the links after each Ingres II upgrade in case libraries were added, removed, or renamed.

UNIX kernel parameters

You should review your UNIX kernel parameter settings, in particular the maximum shared memory size. Ingres II builds a larger locking and logging shared memory segment than Ingres 6.4 did; you will probably need to increase the maximum shared memory segment size. Allowing a 40-Mb shared memory segment will accommodate most migrated installations. Each platform has its own way of modifying the shared memory limits; see your system administrator or your platform specific release notes.

Ingres startup and shutdown

Ingres II uses slightly different startup and shutdown commands (ingstart and ingstop instead of iistartup and iishutdown). If you have shellscripts that start and stop Ingres (perhaps at system boot-up and shutdown time), you will have to change them. Use your development Ingres II installation to verify the changes, and have the revised scripts ready for live upgrade time.

ingprenv replaces ingprenv1

In Ingres 6.4, the ingprenv1 command displayed one Ingres environment variable. This command no longer exists in Ingres II; you use ingprenv instead. You will need to prepare revised versions of any shellscripts that you might have that use ingprenv1, and have them ready for live upgrade time - or, create your own ingprenv1 that just calls ingprenv; for example:

```
echo 'exec $II_SYSTEM/ingres/bin/ingprenv $*'
>/usr/local/bin/ingprenv1

chmod +x /usr/local/bin/ingprenv1
```

System monitoring shellscripts

Many mission critical production systems will have some kind of system monitoring in place to provide the system administrator with an early warning of Ingres problems. If your installation wrote homegrown Ingres monitoring shellscripts, you will have to review them to see if any changes are needed for Ingres II. Some of the sorts of things to watch out for are:

- Checking for iislaves (no more slaves on many platforms).
- Checking for specific lockstat or logstat fields (they may have moved or may look different).
- Checking II_RCP.LOG or II_ACP.LOG (these have been renamed to iircp.log and iiacp.log).
- Checking Ingres 6.4 style parameter files such as rundbms.opt (all Ingres II parameters are in config.dat now).

In addition, you will have to check for issues already mentioned, including the need for LD_LIBRARY_PATH and the use of ingprenv1. If you are using a commercial monitoring system of some sort, contact the vendor to see if you need any updates to support Ingres II.

Checkpoint template changes

The Ingres II checkpoint template file (cktmpl.def) is similar in structure to the Ingres 6.4 version, but it is considerably expanded and not directly compatible. If you have made any changes to your Ingres 6.4 checkpoint template, you will have to recreate those changes for Ingres II. You can use your Ingres II development installation to develop your new cktmpl.def, and it will be ready for your live upgrade. Chapter 17 of the Database Administrator's Guide discusses the new format of the checkpoint template file.

If you have modified your Ingres 6.4 checkpoint template to do parallel checkpointing of multiple locations, be aware that Ingres II supports parallel checkpointing directly; you may be able to simplify your checkpoint processing considerably.

Archiver exit shellscript

Ingres II comes with a sample archiver exit script, called acpexit.def, but does not install it. If you have a custom acpexit from Ingres 6.4, or if you want to install the sample, you have to do this manually after the upgrade. Refer to Chapter 6 of the System Reference Guide for information about the acpexit script.

Transaction log size

Generally, Ingres II does not use as much transaction log space as Ingres 6.4 does. There are a few operations that use more, though (such as MODIFY TO MERGE), and in addition the force-abort limit cannot be set as close to log-full as was possible in Ingres 6.4. If your Ingres 6.4 transaction log was just barely large enough in Ingres 6.4, you may want to expand it somewhat when you move to Ingres II. Your system administrator will appreciate as much warning of this as possible, especially if the log is a raw log.

Backup and Restore

As part of the upgrade, whether done with upgradedb or unload/reload, you will need a system backup. If something goes wrong, you will need to restore that backup. Make sure you or your system administrator knows HOW to take a complete system backup, and how to restore that backup. This sounds crazy to most people; but the panic after a botched upgrade is not the time to be pulling out the man page on your platform's restore command. Nor is it the time to discover that your tape drive can no longer read the tapes it is writing. Check your hardware before you run the live upgrade.

Testing And Practicing

As you make application changes for Ingres II compatibility, you should periodically bring the changes over to the development Ingres II installation and test the application. The amount of testing you do is up to you, but at the very least you should test the mission critical application functions. It would be prudent to test with a reasonable amount of data (a few thousand rows, say, not just one or two).

If testing resources are available, more testing is always better. Since resources are not often available, it becomes more cost effective to plan to react to bugs and incompatibilities instead of trying for 100 percent test coverage. The key word here is *plan*. Have bug-fixing resources in place for several days after the upgrade. Avoid new feature development for those few days. Have modified or short-circuited change control procedures ready so that you can move quickly if a problem crops up.

Make sure you test your system administration procedures, too. At minimum, you should crash your test Ingres II installation when it is busy (pull the power plug, or use kill -9 to crash the servers). Make sure that everything recovers properly. Do at least one rollforwarddb of your most important database(s) and make sure it works in your environment.

When your applications are reasonably Ingres II ready, you can try running an upgrade. Your first upgrade attempt should be done on a test Ingres 6.4 installation; one that can be out of service for a day or two, and ideally one on a separate computer. You will almost certainly want to practice the upgrade more than once, so an isolated environment is desirable.

As you do each practice upgrade, take notes on what went wrong or what you would do differently. Keep doing practice upgrades until you do not take any more notes! Give your annotated upgrade procedure to someone else and let him or her run through an upgrade.

Your first Ingres II upgrade may well be difficult and scary; your third or fourth will be easy. You do not need a complete live dataset to get a valid upgrade practice session. You should however have at least some data, so that you have an idea of how long the upgrade will take.

As you approach the date for the live upgrade, you should delete ALL application objects and images from the development Ingres II installation and re-image a fresh copy of everything, from scratch. Subject this fresh copy to at least a quick critical-functions test. Use this build for your live upgrade.

The Upgradedb Upgrade - A Quick Overview

The upgrade using upgradedb transforms your database in-place from an Ingres 6.4 to an Ingres II format. This is a complex process due to the large number of enhancements made. Upgradedb has presumably been carefully written and tested, and thus most sites ought to be able to use it without any special preparation.

The idea behind the following upgradedb procedure is that the less work upgradedb does, the better. So, each database is prepared by dropping all easily recreate-able objects --essentially, dropping everything but the base tables. Additionally, each base table is checked to make sure it is valid and has no internal damage. After Ingres II is installed and upgradedb run, the various database objects are recreated.

The procedure allows you to cut and paste the output of an unloaddb run to generate SQL that recreates database objects and storage structures. If your site already has canned procedures that recreate database objects and storage structures, feel free to use them instead. Just make sure that your procedures recreate ALL the relevant objects. If users or applications can create new database objects, remember them; you might be better off using unloaddb cutting/pasting in that case.

The upgradedb procedure assumes that you can become any user who owns objects in any database (via login or su). If this is not feasible, you can run as user ingres, and use the -u{user} flag to pretend to be that user any time you have to run an Ingres command.

The Upgradedb Upgrade Procedure

In this procedure, the notation [Each DB] (use standard conventions) means: "For each database, not including the iidbdb: Become the DBA user for that database; cd to the unload directory for that database that you create in step 1; and do this step." Do not include the iidbdb or Star distributed databases in the list unless instructed. If you are using Ingres Star, remember to include the CDB in the list of databases. (The CDB is the coordinator database, the one that usually starts with "ii").

Step 1. [Each DB Including Star DDBs] Create Unload Directories

Create a directory for each database. You will use this directory for holding various scripts (but no data), so you will not need much disk space. A megabyte per directory is generous. Make the directory world writeable.

```
mkdir /someplace/dbname
chmod 777 /someplace/dbname
```

Step 2. [Each DB Including Star DDBs] Run Unloaddb

Run unloaddb against each database. Remember that unloaddb will not unload any data, it just creates copy-in and copy-out scripts. You will take those scripts apart later, giving you a collection of scripts that recreate the various database objects and storage structures. Ingres-STAR note: You do a regular unloaddb of the CDB, but an unloaddb /star of the DDB.

```
unloaddb dbname (regular db or CDB)
unloaddb ddbname/star (STAR distributed db)
```

Steps 2 through 4 can be omitted if you already have your own canned procedures to recreate all database objects and storage structures in all databases. You will also have to make the appropriate changes to oi_prep.sh for remodifying all tables.)

Step 3. [Each DB Including Star DDBs] Bogus User Check

Examine the unload.ing and reload.ing scripts that unloaddb created. Each script contains one line for any user who owns a database object. Make sure that all the users listed are valid; old databases may well have junk objects created by users who are no longer around.

If you find any obsolete users, delete those lines from unload.ing and reload.ing; delete the cp{user}.in and cp{user}.out files; and you might as well go into the database itself and clean out the junk objects.

Step 4. [Each DB] Munge Unloaddb Output

The unloaddb output needs to be modified for recreating just the database objects and storage structures.

Edit each cp{user}.in file EXCEPT the cp_ingre.in file. Starting at the end, you will see a block of CREATE RULE statements; extract them all into one file named {user}_rule.sql. Next are CREATE PROCEDURE statements; extract them all (and their accompanying GRANT statements, if any) into one file named {user}_dbp.sql. Next are CREATE DBEVENT statements and associated GRANTs; extract them into {user}_event.sql. Finally extract all CREATE VIEW, QUEL DEFINE VIEW, and associated GRANTs into {user}_view.sql. This step is best done manually with a text editor.

For each user with a cp{user}.in file, extract all the modify statements into a file, and all modify plus all create index statements into another. This can be done using the commands:

```
sed -n -e '/^modify/,/\\p\\g/p' cp{user}.in
>{user}_modify.sql
```

Use standard CA syntax conventions

```
cp {user}_modify.sql {user}_modindex.sql
sed -n -e '/^create index/,/\\p\\g/p' cp{user}.in
>>{user}_modindex.sql
```

When you are done with this step, you should have SQL scripts that can recreate any database object and storage structure (except base tables) owned by any user in any database.

Step 5. [Each DB Including iidbdb] Optional Checkpoint

Checkpoint each database with ckpdb. This step is optional, since you are going to take a system backup and another checkpoint later. Play it safe and have a valid, recent checkpoint on that backup tape, though. Remember to checkpoint the iidbdb.

Step 6. Disable User Access

From here through the end of the upgrade, you need to turn off user access to any database. How you do this is up to you: prevent user logins, pull the network cable out of the box, whatever works for you.

Step 7. Ingres-Down System Backup

Shut Ingres down. This MUST be a clean shutdown, leaving no information in the Ingres transaction log to be redone or aborted later. One way of doing this is to shut Ingres down. Then start it up and shut it down again. Then check the recovery process log (II_RCP.LOG) for "RCP Shutdown completed normally".

Now, take a system backup, using whatever dump command is appropriate to your platform. You MUST make sure you include all Ingres directories: data, checkpoint, journal, dump areas, and the $II_SYSTEM/ingres directory containing Ingres files and executables. Remember to save your application directories too, as they contain Ingres 6.4 applications. Watch out for symbolic links and cross-mounts, especially in an older installation; make sure you are saving real data and not a symbolic link. Older installations which have been through disk-space panics may well have various Ingres related areas linked to unexpected places.

If you normally start up Ingres at boot time, include the root filesystem in your backups. Or, at least make a manual copy of any Ingres boot time startup and shutdown scripts.

To play it safe, do this step in single-user mode, after cleaning the tape drive. Using brand new, never-used tapes. Do this twice and check the tapes after backups to insure that you can read them.

After your backup is finished, start up Ingres again.

Step 8. [Each DB] Optional Statdump

Only do this step if you are pressed for time and can not run a full optimizedb run against the database when the upgrade is done. You can dump out the existing optimizer statistics and reload them after the upgrade. You will not get some of the new Ingres II metrics this way, but it is better than nothing. If you have enough time to run optimizedb against your databases, it is preferred that you omit this step, and run the optimizedb.

Run statdump with the -o flag to a file for each database:

```
statdump -o dbname.stats dbname
```

Step 9. [Each DB] Object Cleaning

Drop all non-table objects from the database: optimizer statistics, views, rules, database procedures, and dbevents. In addition, re-modify all tables to verify their validity, and run some verifydb checks against the database. You can use the shellscript provided at the end of this Guide (see oi_prep.sh) to do the work automatically. Using the C-shell:

```
oi_prep.sh dbname |& tee oi_prep.log
```

If your database has dependent views, you will probably see some DROP errors on those views. (oi_prep.sh does not bother to drop views in reverse dependency order.) Ignore those DROP errors.

The verifydb -odbms command will very likely output a bunch of messages of the forms:

S_DU1611_NO_PROTECTS iirelation indicates that there are protections for table (owner), but none are defined.

S_DU0305_CLEAR_PRTUPS Recommended action is to clear protection information from iirelation, and S_DU1619_NO_VIEW iirelation indicates that there is a view defined for table (owner), but none exists.

S_DU030C_CLEAR_VBASE Recommended action is to clear view base specification from iirelation.

You may safely ignore both of these kinds of messages. Also ignore the "patch warning" message that warns of the loss of user tables in the RUNINTERACTIVE mode as you are not using this mode.

Some databases (especially older ones) may produce a bunch of messages from verifydb, then output a "verifydb failed" message with no explanation and exit. If this happens, run the Terminal Monitor with the update system catalogs flag:

```
sql +U dbname
SELECT * FROM iistatistics;\go
```

You should not see any rows. If you do, they are likely the cause of the verifydb problem. Simply delete them:

```
DELETE FROM iistatistics;COMMIT;\go
\quit
```

Re-run the verifydb command at the end of oi_prep.sh and make sure you do not get any errors.

If you see any other error messages from verifydb, you will have to stop and correct the system catalog problems before continuing. Contact Tech Support for help if necessary.

Do not process Ingres Star distributed databases. These should be upgraded as-is.

Step 10. [Each DB] Record INFODB

Run infodb against each database, saving the output. You will want this later mostly so that you know whether the database was journaled or not, and if something goes wrong you'll know what data locations the database lives in, and what order they are in.

```
infodb dbname >infodb.out
```

Step 11. Clean iidbdb

Become user ingres, and run a subset of the Object Cleaning step against the master database iidbdb. We'll assume that there are no user created objects in the iidbdb to deal with (there shouldn't be!).

```
statdump '-u$ingres' -zdl iidbdb
sysmod -s iidbdb
verifydb -mrun -sdbname iidbdb -opurge
verifydb -mrun -sdbname iidbdb -odbms
```

You may get some warnings from verifydb that should be ignored. See Step 9 (Object Cleaning) for details.

Step 12. [Each DB Including iidbdb] Checkpoint -j

Checkpoint each database now that they have been cleaned up. You are doing this for two reasons: one, to turn off journaling for all databases; and two, because if upgradedb has a hiccup, you can use this checkpoint to recover relatively quickly and try again.

Checkpoint TIP

YOU SHOULD DO THIS FOR IIDBDB AS WELL. YOU DO NOT HAVE AN "UNLOAD" DIRECTORY FOR IIDBDB, SO JUST STORE ITS AAAAAAAA.CNF FILE ANYWHERE SAFE.

In addition to checkpointing, you should save the config file stored in the dump area after each ***checkpoint***. Config files are small, and the more data you save, the more recovery options you have if something goes wrong.

```
ckpdb -d -j dbname
cp {dump
location}/ingres/dmp/default/{dbname}/aaaaaaaa.cnf .
```

Step 13. Record Ingres Configuration

As user ingres, do a "showrcp" command and record the results. Also, record the contents of the rundbms.opt file in $II_SYSTEM/ingres/files. You will use this information as a rough guideline for configuring Ingres II. The Ingres II installation procedure does not preserve your existing Ingres tuning parameter settings, which is why you should record your current values now. You are going to delete the ingres/files directory later, so record this someplace safe.

Step 14. Ingres 6.4 Shutdown

Shut down ingres using iishutdown.

Step 15. Disable Ingres Startup

If your machine normally starts Ingres automatically when booting, turn auto-starting off to avoid auto-starts before you are ready.

On most UNIX platforms, there will be a file in /etc/init.d or /sbin/init.d that does Ingres startup and shutdown; just put an "exit 0" at the top of that file. You may have to be root or have your system administrator do this step.

While you have the system administrator around as a resource, make sure that your operating system is correctly configured for Ingres II. (See "System Administration Preparation", page 16.) If you need to reboot to (say) increase the shared memory limit, you need to do this at this time.

Step 16. Preserve Site Modifications

Many sites install a variety of customizations in their $II_SYSTEM directory tree. The most common customizations are added or changed termcap and keyboard map files in $II_SYSTEM/ingres/files. You may also have customizations in the bin or utility directory. Remember to check for local collation sequence files. Ideally, save the original collation definition files; but you should save the compiled files in $II_SYSTEM/ingres/files/collation as well.

Copy any customized files to a safe place, meaning NOT /tmp and NOT anywhere underneath the $II_SYSTEM/ingres directory. If you are not entirely positive that you can identify all customized files, do this: delete all *.log and *.LOG files from $II_SYSTEM/ingres/files. Then, copy the entire contents of the $II_SYSTEM/ingres/bin, $II_SYSTEM/ingres/files, and $II_SYSTEM/ingres/utility directories somewhere safe. This is copying more than needed, but you can always delete your copy later. You may not discover you need some strange Vision template or keyboard map for weeks, as parts of the $II_SYSTEM/ingres directory tree permanently.

If you choose this copy method, this tar command will copy everything you are likely to need:

```
cd $II_SYSTEM/ingres
tar cf - bin files utility | (cd /someplace/safe;tar
xf -)
```

Step 17. Login Fixups

Make sure that the ingres user login sets LD_LIBRARY_PATH (or your platform's equivalent) if needed, and make sure that it does not use ingprenv1 (or install your ingprenv1 substitute). See "System Administration Preparation", page 16. You may want to check all your database owner (DBA) logins at this point as well to ensure that all users are properly set up for Ingres II.

Step 18. Record Default Locations

The upgrade runs the smoothest if you delete away the Ingres 6.4 executables and control files, including the Ingres environment variables. This means that you will have to re-enter your default Ingres locations, so make sure you know what they are. Do "ingprenv" and record the values of II_DATABASE, II_CHECKPOINT, II_JOURNAL, and II_DUMP.

Step 19. Clean off Ingres 6.4

Remove the Ingres 6.4 bin, files, lib, and utility directories. This gives you a fresh start for Ingres II.

```
cd $II_SYSTEM/ingres
rm -rf bin files lib utility dbtmplt version.rel
admin
```

Step 20. Create Work Location

Ingres II asks you to create an Ingres location for temporary files and sorting. The installation procedure will create the directories for you. However, in some situations, the installation procedure may not properly protect the directories, leading to upgradedb failure on iidbdb. To avoid this, you may want to consider creating the work location by hand. Refer to the Database Administrator's Guide, Chapter 5, "Using Work Locations" for information on placement of your default work location. As user ingres, assuming a work location of /work:

```
mkdir /work/ingres /work/ingres/work
mkdir /work/ingres/work/default
/work/ingres/work/default/iidbdb
chmod 755 /work/ingres
chmod 700 /work/ingres/work
chmod 777 /work/ingres/work/default
chmod 777 /work/ingres/work/default/iidbdb
```

Step 21. Install Ingres II

Refer to the Ingres II installation instructions for your platform. Enter the directory locations that you recorded in Step 18 (Record Default Locations) for the default database areas (ii_database, ii_checkpoint, ii_journal, and ii_dump). During the DBMS server setup, it will ask you if you want to upgrade all your databases; answer No.

The install procedure will upgradedb the iidbdb anyway; if this fails you probably have something substantially wrong (See "Upgradedb Problems", page 35). It is best to get Ingres II entirely set up, then run through the upgradedb's on the user databases.

If you plan on installing a patch to Ingres II, you should say NO when ingbuild asks you whether you want to set up Ingres II. Instead, exit ingbuild. Install the patch using the patch instructions. Then, re-run ingbuild. Select Current, then SetupAll. By doing this, if there is a fix to upgradedb or the installation setup in the patch, you will setup with the fixed version and not the original.

Step 22. Restore Site Modifications

If your site modified the checkpoint template cktmpl.def, you need to recreate your modifications for Ingres II. You can not use the cktmpl.def from Ingres 6.4, as the file format has been expanded in Ingres II. You will have to start over, using your Ingres 6.4 modifications as a guide. Your Database Administrator's Guide has a section on cktmpl.def.

Likewise, if your site uses the archiver exit script acpexit, you need to modify the template shipped with Ingres II (acpexit.def), and install it as acpexit in $II_SYSTEM/ingres/files.

Go to your safe place from Step 16 (Preserve Site Modifications), and restore any files that are specific to your site. In particular, make sure that you restore any local collation sequence files to $II_SYSTEM/ingres/files/collation. Re-run aducompile on the sequence definitions if you have them; if not, you can reuse the compiled collation sequences from Ingres 6.4 .

Step 23. Start Ingres II

The installation procedure generally leaves the Ingres II servers stopped, so start them again. Remember that the startup command is "ingstart" in Ingres II, not "iistartup".

Step 24. Upgradedb

As user ingres, run the command "upgradedb -all" to upgrade all database. If you have followed the procedure so far, this step should run without any problems. Refer to the "Upgradedb Problems" section if something does go wrong; it is very likely that you will be able to fix the problem and keep going. It's a good idea to log upgradedb's output to a file; in the C Shell:

```
upgradedb -all |& tee upgradedb.log
```

If you make it through this step successfully, relax; the hard part is over, and you are almost certain to complete the rest of the upgrade without any problems.

Step 25. Configure Ingres II

Run CBF (Configuration-By-Forms) and do a first-cut configuration of your Ingres II installation. Use the rundbms.opt and showrcp information you recorded earlier as a guideline. Do not worry about getting things exactly right the first time; you are mainly trying to get a reasonable configuration. Refer to your System Reference Guide for information about CBF and the various tuning parameters.

TIP 1: *DERIVED PARAMETERS ARE RECALCULATED WHEN VALUES THEY DEPEND ON ARE CHANGED. IF YOU SET A DERIVED PARAMETER, YOU MAY WANT TO "PROTECT" IT AGAINST BEING UNEXPECTEDLY CHANGED.*

TIP2: *INGRES II TENDS TO CALCULATE VERY LARGE LOCK LIMIT AND RESOURCE LIMIT DERIVED PARAMETERS. THIS IS APPROPRIATE FOR SITES USING ROW-LEVEL LOCKING, BUT MAY BE EXCESSIVE FOR UPGRADED SITES. CONSIDER CUTTING THESE LIMITS BACK CLOSE TO INGRES 6.4 LEVELS.*

TIP 3: *ON OS-THREADS PLATFORMS, DO NOT TURN ON ASYNC_IO; AND DO NOT DECLARE THE II_NUM_SLAVES INGRES VARIABLE.*

TIP 4: *INGRES II CAN SUPPORT LARGER QEF_SORT_MEM VALUES THAN INGRES 6.4. INGRES II DOES NOT NEED AS MUCH QSF_MEMORY AS INGRES 6.4 DID. OS-THREAD PLATFORMS SHOULD NOT REDUCE QUANTUM; UNLIKE INGRES 6.4, REDUCING QUANTUM ON OS-THREAD PLATFORMS WILL NOT IMPROVE RESPONSIVENESS.*

Step 26. Ingres-Net Setup

If you are accessing Ingres installations remote to this site, run netutil to recreate the vnode definitions for those remote installations. If you have NFS client-only installations that you have not set up yet, run ingmknfs to set them up.

If your platform needs "ingvalidpw" (the setuid-root password checker program), you should re-create it now by running mkvalidpw. Refer to the System Reference Guide, or your platform release notes. You can defer this step until later, unless you are running Ingres Star.

Step 27. [Each DB] Recreate Objects

For each {user}_view.sql script generated by Step 4 (Munge Unloaddb Output), recreate those views:

```
sql -u{user} dbname <{user}_view.sql
```

In the same way, recreate all dbevents, database procedures, and rules in that order.

Step 28. [Each DB] Reapply Storage Structures

For each {user}_modindex.sql script generated by Step 4 (Munge Unloaddb Output), reapply storage structures and indexes:

```
sql -u{user} dbname <{user}_modindex.sql
```

It is essential to redo secondary indexes, since they were dropped by the oi_prep.sh preparation. It's highly desirable, but not absolutely essential, to redo all the table modifys as well. If you are extremely short of time, you can omit the modifys and just do the create index statements. (You can also consider dividing up the input {user}_modindex.sql files, and running multiple sessions at once. Even a uniprocessor system can benefit from running two modify sessions simultaneously.)

Step 29. [Each DB] Reapply Optimizer Statistics

Run whatever your normal procedure is for generating optimizer statistics with the optimizedb command. If you are short of time, and you dumped out your Ingres 6.4 statistics in Step 8 (Optional Statdump), read the Ingres 6.4 statistics back in:

```
optimizedb dbname -i dbname.stats
```

Ingres II computes some additional statistics for better query optimization, so it is better to re-run a regular optimizedb. Ingres 6.4 statistics are better than nothing.

Step 30. [Each DB including iidbdb] Checkpoint

Checkpoint each database, using the +j flag to turn on journaling if the database was journaled before. (Refer to the infodb output from Step 10 to see which databases were journaled).

ALWAYS turn on journaling for the master database, iidbdb.

Step 31. Application Upgrade

Install the Ingres II versions of all your applications, using whatever procedure is normal for your site. Then, restore user logins, and resume normal operation.

This completes the upgradedb upgrade procedure.

Upgradedb Problems

The upgradedb program is complex, and has been afflicted some peculiarities that have become rather legendary, and have led to some sites doing unnecessary unload/reload work just to avoid upgradedb.

The procedure given above will avoid most upgradedb problems, whether real or mythical. Here is a list of problems that you might see even if you follow the procedure, and how to recover from them. Caveat: the problems in this list are the result of many upgrades worth of experience. Not all problems exist in all versions, as CA is naturally working to resolve as many deficiencies as possible. Non-critical upgradedb problems tend to languish behind more critical issues, though, which is why some of these bugs have been around for a while.

Problem 1: "Can't delete database from server."

Upgradedb will sometimes start to upgrade a database, then quit after a few Converting and Upgrading messages with: "Can't delete database from server." This will only happen to some databases in an installation, with no apparent pattern.

This problem was seen in OpenIngres 1.2. The fix is simply to re-issue the command "upgradedb -all". Apparently no damage is done to the databases not upgraded, and trying again usually works.

Problem 2: "Duplicate Key" upgrading iidbdb

If upgradedb is re-run after iidbdb is already upgraded, a "Duplicate key on insert" message appears, followed by scary looking warnings about iidbdb. The upgradedb then continues normally.

This message is related to upgradefe, which upgrades the front-end catalogs. The "fix" is to simply ignore the message. No harm is done, and this problem is corrected in the 9712 genlevel of OpenIngres 2.0.

Problem 3: "Product .. has been made uninstallable."

Upgradedb will occasionally print a message "Product <name> has been made uninstallable by an incompatible dictionary upgrade."

This message is actually related to upgradefe, which upgrades the front-end catalogs, and this message can be safely ignored. The message seems to be provoked by some older databases that may have gone through a failed product installation at some time in the past.

Problem 4: Hang with open() error 13

Upgradedb will start upgrading the iidbdb, then hang part way through with no obvious message. The ingres error log $II_SYSTEM/ingres/files/errlog.log shows a message "open() error 13".

This problem is caused by an error in the permissions on the work directory structure (the iidbdb work directory gets protected 664 rather than 777). The upgradedb procedure given above avoids this bug by pre-creating the affected directory. Other databases have the work directory created properly. The problem may be caused by the ingres user's .cshrc file not containing a "umask 0" command, as is recommended by the Getting Started installation manual.)

Problem 5: File extend conversion loop

Upgradedb will loop printing "file extend converting {table name}" over and over.

This problem was observed when a damaged database had a system catalog entry for a table, but the underlying file was missing. (The actual error can be seen in the Ingres error log, errlog.log.) The upgradedb procedure avoids this problem by remodifying all tables (broken tables will produce an error during the modify, and the condition can be corrected). If the situation occurs, it can be rectified by aborting upgradedb; shutting down ingres; manually copying any valid not-yet-upgraded table file to the missing filename, which you get from errlog.log; starting up ingres; and rerunning upgradedb.

Problem 6: "Failed, aborting" creating internal procedure

If upgradedb is aborted (perhaps by a system crash) during processing, re-running upgradedb may produce the normal message "creating internal procedure iiqef_alter_extension", followed by a "failed, aborting" message. This will not happen all the time- only when upgradedb is interrupted when it is nearly completed.

This problem appears to only be present in the 9712 genlevel of OpenIngres 2.0. It is caused by a failure to detect that internal procedure "iierror" already existed from the first, aborted run. The fix is available from CA technical support.

The only known workaround is to restore the database from the checkpoint taken in Step 12, and re-run the upgrade. Shut down Ingres; delete the files from each data location of the database; un-tar each checkpoint file(s) into its corresponding data location; copy the aaaaaaaa.cnf file that you preserved, into the root data location and the dump directory; restart Ingres; and redo the upgradedb. Use the infodb output as a guide to which checkpoint files go with what data locations. (If you are not used to working with checkpoint files, either get help from Tech Support, or abort the entire upgrade; restore from backup; and start over.)

Problem 7: iifile_info view not recreated

Upgradedb drops the iifile_info system catalog view but does not recreate it.

This problem appears to only be present in the 9712 genlevel of Ingres II. It is fixed (via patch) on some platforms; check with Tech Support. The workaround is to create the view manually:

```
sql '-u$ingres' +U dbname

----- iifile_info view definition (2.0/9712) ----
create view iifile_info(table_name, owner_name,
file_name, file_ext,
location, base_id, index_id)as select r.relid,
r.relowner,
ii_tabid_di(reltid, reltidx), 't00', r.relloc,
r.reltid, r.reltidx
```

```
from "$ingres". iirelation r where r.reltidx=0
and(mod((r.relstat/32),
(2))=0)union all select r.relid, r.relowner,
ii_tabid_di(reltid,
reltidx), 't00', r.relloc, r.reltid, r.reltidx from
"$ingres".
iirelation r where r.reltidx!=0 union all select
r.relid, r.relowner,
ii_tabid_di(reltid, reltidx), 't'
+charextract('0123456789abcdef',
mod((d.devrelocid/16), (16)) +1)
+charextract('0123456789abcdef',
mod((d.devrelocid), (16)) +1), d.devloc, r.reltid,
r.reltidx from
"$ingres". iirelation r, "$ingres". iidevices d where
r.reltidx=0
and(mod((r.relstat/32), (2))=0)and
d.devrelid=r.reltid and
d.devrelidx=r.reltidx and(mod((r.relstat/4194304),
(2))!=0)union all
select r.relid, r.relowner, ii_tabid_di(reltid,
reltidx), 't'
+charextract('0123456789abcdef',
mod((d.devrelocid/16), (16)) +1)
+charextract('0123456789abcdef', mod((d.devrelocid),
(16)) +1),
d.devloc, r.reltid, r.reltidx from "$ingres".
iirelation r, "$ingres".
iidevices d where r.reltidx!=0
and(mod((r.relstat/4194304), (2))!=0)
and d.devrelid=r.reltid and d.devrelidx=r.reltidx
--------- end of iifile_info view definition --------
-
```

Problem 8: ALL to PUBLIC grants not created

Upgradedb preserves the ALL-to-ALL flag in iirelation that allows ALL to PUBLIC access; but it neglects to create the individual Ingres II-style permits in the iipermit table. The symptom is that the affected tables are not visible in the QBF table listing except to the owner.

This problem was present in OpenIngres 1.2; it is not confirmed in Ingres II. The cure is to simply re-issue the GRANT ALL to PUBLIC statements at some convenient time after the upgrade is complete.

The Unload/Reload Upgrade: Overview

The unload/reload upgrade avoids the upgradedb program (except for iidbdb), in favor of dumping the Ingres 6.4 database(s) to flat files, recreating the databases under Ingres II, and reloading them. This approach has the rather chimerical advantage of starting your Ingres II installation out with all new, fresh databases. The downside is that substantially more time and disk space will be needed.

The unload/reload upgrade is slightly simpler than the upgradedb upgrade, since there is no need to defend against real or imagined upgradedb problems. The trickiest part of the unload/reload upgrade is dealing with the front-end catalogs. These are dumped in Ingres 6.4 format, and cannot be loaded as is into an Ingres II database. So, the Ingres II database is created without front-end catalogs. The catalogs are reloaded in the Ingres 6.4 format, and upgradedb using the upgradefe program.

The iidbdb is not unloaded and reloaded. That would be tantamount to a cold install of the entire installation. Instead, iidbdb is upgraded with upgradedb. This way, your existing users, locations, groups, roles, etc. are preserved, and need not be re-entered.

The Unload/Reload Upgrade

In this procedure, the notation [Each DB] means: "For each database, not including the iidbdb: Become the DBA user for that database; CD to the unload directory for that database that you created in step 1; and do this step." If you are using Ingres Star, remember to include the CDB in the list of databases. (The CDB is the coordinator database, the one that usually starts with "ii".)

Step 1. [Each DB] Create Unload Directories

Create a directory for each database. This directory will be used for holding various scripts the unloaded database, so you'll need plenty of disk space. Usually the unloaded data is about the same size or a bit smaller than the Ingres database; however compressed data can "blow up" and take up much more room than the Ingres database. Make the directory world writable.

```
mkdir /someplace/dbname
chmod 777 /someplace/dbname
```

Step 2. [Each DB] Run Unloaddb

Run unloaddb against each database. Remember that unloaddb do not actually unload any data, it just creates copy-in and copy-out scripts. You will run these scripts later. Ingres Star Note: You do a regular unloaddb of the CDB, but an unloaddb /star of the DDB.

```
unloaddb dbname (regular db or CDB)
unloaddb ddbname/star (STAR distributed db)
```

Step 3. [Each DB] Bogus User Check

Examine the unload.ing and reload.ing scripts that unloaddb created. Each script contains one line for any user who owns a database object. Make sure that all the users listed are valid; old databases may well have junk objects created by users who are no longer around.

If you find any obsolete users, delete those lines from unload.ing and reload.ing; delete the cp{user}.in and cp{user}.out files; and you might as well go into the database itself and clean out the junk objects.

Step 4. [Each DB Including iidbdb] Optional Checkpoint

Checkpoint each database with ckpdb. This step is optional, as you are going to take a system backup soon. It is a good idea to have a valid, recent checkpoint on that backup tape. Remember to checkpoint the iidbdb.

Step 5. Disable User Access

From here through the end of the upgrade, you need to turn off user access to any database. How you do this is up to you - prevent user logins, pull the network cable out of the box, whatever.

Step 6. Ingres-Down System Backup

Shut Ingres down. This MUST be a clean shutdown, leaving no information in the Ingres transaction log to be redone or aborted later. One way of doing this is to shut Ingres down. Then start it up and shut it down again. Then check the recovery process log (II_RCP.LOG) for "RCP Shutdown completed normally".

Now, take a system backup, using whatever dump command is appropriate to your platform. You MUST make sure you include all Ingres directories: data, checkpoint, journal, dump areas, and the $II_SYSTEM/ingres directory containing Ingres files and executables. Remember to save your application directories too, as they contain Ingres 6.4 applications. Watch out for symbolic links and cross-mounts, especially in an older installation; make sure you are saving real data and not a symbolic link. Older installations which have been through disk-space panics may well have various Ingres related areas linked to unexpected places.

If you normally start up Ingres at boot time, include the root filesystem in your backups. Or, at least make a manual copy of any Ingres boot time startup and shutdown scripts.

It is a good idea, if possible, to do this step in single-user mode, after cleaning the tape drive. Using brand new, never-used tapes. Do this twice and check both tapes for validity prior to proceeding.

After your backup is finished, start up Ingres again.

Step 7. [Each DB] Unload

For each database, run the "unload.ing" script created by unloaddb. This will actually unload the database data into your unload directory.

Step 8. [Each DB] Optional Statdump

Only do this step if you can't afford the time to run a full optimizedb run against the database when the upgrade is done. You can dump out the existing optimizer statistics and reload them after the upgrade. You won't get some of the new Ingres II metrics this way, but it's better than nothing. If you have enough time to run optimizedb against your databases, it's preferred that you omit this step. Run statdump with the -o flag to a file for each database:

```
statdump -o dbname.stats dbname
```

Step 9. [Each DB] Record INFODB

Run infodb against each database, saving the output. You'll need this later so that you can extend the re-created database to its original locations. You'll also want to know if the database was originally journaled or not.

```
infodb dbname >infodb.out
```

Step 10. [Each DB] Destroy the database

Destroy each database using destroydb.

Step 11. Clean iidbdb

Become user ingres, and run a subset of the upgradedb's Object Cleaning step against the master database iidbdb. We will assume that there are no user created objects in the iidbdb to deal with (there shouldn't be!).

```
statdump '-u$ingres' -zdl iidbdb
sysmod -s iidbdb
verifydb -mrun -sdbname iidbdb -opurge
verifydb -mrun -sdbname iidbdb -odbms
ckpdb -d -j -s iidbdb
```

You may get some warnings from verifydb that should be ignored. See upgradedb Step 9 (Object Cleaning) for details.

Step 12. Record Ingres Configuration

As user ingres, do a "showrcp" command and record the results. Also, record the contents of the rundbms.opt file in $II_SYSTEM/ingres/files. You will use this information as a rough guideline for configuring Ingres II. The Ingres II installation procedure does not preserve your existing Ingres tuning parameter settings, which is why you should record your current values now.

Step 13. Ingres 6.4 Shutdown

Shut down ingres using iishutdown.

Step 14. Disable Ingres Startup

If your machine normally starts Ingres automatically at boot time, you should turn auto-starting off. You do not want an unfortunately timed system crash to try to start Ingres when it is partially upgraded.

On most UNIX platforms, there will be a file in /etc/init.d or /sbin/init.d that does Ingres startup and shutdown; just put an "exit 0" at the top of that file. You will probably have to be root or have your system administrator do this step.

While you have the system administrator around, make sure that your operating system is configured for Ingres II. (See "System Administration Preparation", page 16) If you need to reboot to increase the shared memory limit, do it now.

Step 15. Preserve Site Modifications

Many sites install a variety of customizations in their $II_SYSTEM directory tree. The most common customizations are added or changed termcap and keyboard map files in $II_SYSTEM/ingres/files. You may also have customizations in the bin or utility directory. Remember to check for local collation sequence files. Ideally, save the original collation definition files; but you should save the compiled files in $II_SYSTEM/ingres/files/collation as well.

Copy any customized files to a safe place, meaning NOT /tmp and NOT anywhere underneath the $II_SYSTEM/ingres directory. If you are not entirely positive that you can identify all customized files, do this:

Delete all *.log and *.LOG files from $II_SYSTEM/ingres/files. Then, copy the entire contents of the $II_SYSTEM/ingres/bin, $II_SYSTEM/ingres/files, and $II_SYSTEM/ingres/utility directories somewhere safe. This is copying more than is necessary, but you can always delete your copy later. You may not discover you need some strange Vision template or keyboard map for weeks, so it is a good idea to archive off the non-database parts of the $II_SYSTEM/ingres directory tree permanently.)

If you choose this copy method, this tar command will copy everything you are likely to need:

```
cd $II_SYSTEM/ingres
tar cf - bin files utility | (cd /someplace/safe;tar
xf -)
```

Step 16. Login Fixups

Make sure that the ingres user login sets LD_LIBRARY_PATH (or your platform's equivalent) if it is needed, and make sure that it does not use ingprenv1 (or install your ingprenv1 substitute). If you have not done this ahead of time, you should do it now. (See "System Administration Preparation", page 16).

You may as well check all your database owner (DBA) logins now too. Make sure that all users are properly set up for Ingres II.

Step 17. Record Default Locations

The upgrade runs the smoothest if you delete away the Ingres 6.4 executables and control files, including the Ingres environment variables. This means that you'll have to re-enter your default Ingres locations, so you had better know what they are. Do "ingprenv" and record the values of II_DATABASE, II_CHECKPOINT, II_JOURNAL, and II_DUMP.

Step 18. Clean off Ingres 6.4

Remove the Ingres 6.4 bin, files, lib, and utility directories. This gives you a fresh start for Ingres II.

```
cd $II_SYSTEM/ingres
rm -rf bin files lib utility dbtmplt version.rel
admin
```

Step 19. Create Work Location.

Ingres II asks you to create an Ingres location for temporary files and sorting. The installation procedure will create the directories for you. However, in certain situations (not isolated as yet!), the installation procedure does not properly protect the directories, leading to upgradedb failure on iidbdb. This is a nuisance, so you might as well create the work location by hand. Refer to the Database Administrator's Guide, Chapter 4, "Using Work Locations" for information on placement of your default work location. As user ingres, assuming a work location of /work:

```
mkdir /work/ingres /work/ingres/work
mkdir /work/ingres/work/default
```

```
work/ingres/work/default/iidbdb
chmod 755 /work/ingres
chmod 700 /work/ingres/work
chmod 777 /work/ingres/work/default
chmod 777 /work/ingres/work/default/iidbdb
```

Step 20. Install Ingres II

Use the Ingres II installation instructions for your platform and install Ingres II. Enter the directory locations that you recorded in Step 17 (Record Default Locations) for the default database areas (ii_database, ii_checkpoint, ii_journal, and ii_dump). During the DBMS server setup, it will ask you if you want to upgrade all your databases; answer No.

The install procedure will upgradedb the iidbdb anyway; if this fails you probably have something substantially wrong (but see "Upgradedb Problems", above).

If you plan on installing a patch to Ingres II, you should say NO when ingbuild asks you whether you want to set up Ingres II. Instead, exit ingbuild. Install the patch using the patch instructions. Then, re-run ingbuild. Select Current, then SetupAll. By doing this, if there is a fix to upgradedb or the installation setup in the patch, you will setup with the fixed version and not the original.

Step 21. Restore Site Modifications

If your site modified the checkpoint template cktmpl.def, you need to recreate your modifications for Ingres II. You can't use the cktmpl.def from Ingres 6.4, as the file format has been expanded in Ingres II. You will have to start over, using your Ingres 6.4 modifications as a guide. The Database Administrator's Guide has a section on cktmpl.def.

Likewise, if your site uses the archiver exit script acpexit, you need to modify the template shipped with Ingres II (acpexit.def), and install it as acpexit in $II_SYSTEM/ingres/files.

Go to your safe place from Step 15 (Preserve Site Modifications), and restore any files that are specific to your site. In particular, make sure that you restore any local collation sequence files to $II_SYSTEM/ingres/files/collation. Re-run aducompile on the sequence definitions if you have them; if not, you can reuse the compiled collation sequences from Ingres 6.4.

Step 22. Configure Ingres II

Run CBF (Configuration-By-Forms) and do a first-cut configuration of your Ingres II installation. Use the rundbms.opt and showrcp information you recorded earlier as a guideline. Do not worry about getting things exactly right the first time; you are mainly trying to get a reasonable configuration. Refer to your System Reference Guide for information about CBF and the various tuning parameters.

TIP 1: *DERIVED PARAMETERS ARE RECALCULATED WHEN VALUES THEY DEPEND ON ARE CHANGED. IF YOU SET A DERIVED PARAMETER, YOU MAY WANT TO "PROTECT" IT AGAINST BEING UNEXPECTEDLY CHANGED.*

TIP2: *INGRES II TENDS TO CALCULATE VERY LARGE LOCK LIMIT AND RESOURCE LIMIT DERIVED PARAMETERS. THIS IS APPROPRIATE FOR SITES USING ROW-LEVEL LOCKING, BUT MAY BE EXCESSIVE FOR UPGRADED SITES. CONSIDER CUTTING THESE LIMITS BACK CLOSE TO INGRES 6.4 LEVELS.*

TIP 3: *ON OS-THREADS PLATFORMS, DO NOT TURN ON ASYNC_IO; AND DO NOT DECLARE THE II_NUM_SLAVES INGRES VARIABLE.*

TIP 4: *INGRES II CAN SUPPORT LARGER QEF_SORT_MEM VALUES THAN INGRES 6.4. INGRES II DOES NOT NEED AS MUCH QSF_MEMORY AS INGRES 6.4 DID. OS-THREAD PLATFORMS SHOULD NOT REDUCE QUANTUM; UNLIKE INGRES 6.4, REDUCING QUANTUM ON OS-THREAD PLATFORMS WILL NOT IMPROVE RESPONSIVENESS.*

.

Step 23. Ingres Net Setup

If you are accessing Ingres installations remote to this site, run netutil to recreate the vnode definitions for those remote installations. If you have NFS client-only installations that you have not set up yet, run ingmknfs to set them up.

If your platform needs "ingvalidpw" (the setuid-root password checker program), you should re-create it now by running mkvalidpw. Refer to your System Reference Guide or your platform release notes. You can defer this step until later, unless you are running Ingres Star.

Step 24. Start Ingres II

The installation procedure generally leaves the Ingres II servers stopped, so start them again. Remember that the startup command is "ingstart" in Ingres II, not "iistartup".

Step 25. [Each DB] Recreate Databases

Recreate each user database, omitting the front-end catalogs. (The front-end catalogs will be reloaded, then upgraded, later on.) Before creating each database, refer to the infodb output saved in Step 9 (Record Infodb); if the "ROOT" data location is not ii_database, you have to specify it in the createdb command:

```
createdb dbname -dlocation -f nofeclients
```

(Remember you specify the location name, not the directory path.)

If the "ROOT" data location is ii_database, you omit the -d option.

Star note: Do not run createdb on the CDB. Instead, run createdb /star on the DDB.

Step 26. [Each DB] Extend Databases

Refer to the infodb output saved in Step 9 (Record Infodb). If the database was extended to any locations other than the default locations, run accessdb now (as user ingres) and extend the newly created databases to the same locations. The locations will already exist, you just need to extend the databases to use them.

Step 27. [Each DB] Fix FE Reload Script

Edit the file cp_ingre.in and locate the line that looks like:

```
\include /ing64/ingres/files/iiud64.scr
```

(Your directory path will vary.) Delete this line and save cp_ingre.in. Since you did not create the database with front-end catalogs, you don't need to drop them, which is what iiud64 attempts to do.

Step 28. [Each DB] Reload

Run reload.ing for each database. It's prudent to tee the reload to a log file so that you can check for errors later. Using the C-Shell:

```
reload.ing |& tee reload.log
```

Star note: Reload the CDB and all "real" local databases first. Then, reload the DDB's.

Step 29. [Each DB] Front-End Upgrade

Run upgradefe on each database, bringing the front-end catalogs up to Ingres II level.

```
upgradefe dbname INGRES
```

The word INGRES should be supplied exactly as shown.

Step 30. [Each DB] Reapply Optimizer Statistics

Run whatever your normal procedure is for generating optimizer statistics with the optimizedb command.

If you are short of time, and you dumped out your Ingres 6.4 statistics in Step 8 (Optional Statdump), read the Ingres 6.4 statistics back in:

```
optimizedb dbname -i dbname.stats
```

Ingres II computes some additional statistics for better query optimization, so it's better to re-run a regular 2.0 optimizedb. Ingres 6.4 statistics are better than nothing, though.

Checkpoint TIP

ALWAYS TURN ON JOURNALING FOR THE MASTER DATABASE, IIDBDB.

Step 31. [Each DB including iidbdb] Checkpoint

Checkpoint each database, using the +j flag to turn on journaling if the database was journaled before. (Refer to the infodb output from Step 9 to see which databases were journaled).

Step 32. Application Upgrade

Install the Ingres II versions of all your applications, using whatever procedure is normal for your site. Then, restore user logins, and resume normal operation.

This completes the unload/reload upgrade procedure.

oi_prep.sh shellscript

This attachment is the oi_prep.sh shellscript mentioned in the upgradedb procedure, step 9 (Object Cleaning).

```
-------------- start of oi_prep.sh ------------------
-

#!/bin/sh
# Prepare for Ingres 6.4 -> Ingres II update.
# Usage: oi_prep.sh dbname
# This script implements Step 9 ("Object Cleaning")
of the upgradedb procedure.
# You run it in the "unload" work directory that
contains the extracts of the unloaddb for the
database. It should be run as the DBA user for the
database.
# Its job is to prep a database for the Ingres II
update, by dropping all nonessential database objects
and running some verification procedures. The
following is done for the database:
# - Drop all statistics
# - Drop all views, procedures, rules, and dbevents
for all users.
# - Reapply all storage structures.
# - Sysmod the database
# - Run verifydb -odbms_check to double-check the
system catalogs.
# - Run infodb and make sure the header says VALID.
The idea is that the less work upgradedb has to do,
the more likely it is that it will do it right!
This script expects to see files named according to
the naming conventions given in the upgradedb
procedure section. It should not be too hard to adapt
to different circumstances if necessary.
Remember the database name:
dbName=$1
if [ -z "$dbName" ] ; then
echo "Usage: oi_prep.sh dbname"
exit 1
```

```
fi
User has to have II_SYSTEM defined, and Ingres must
be up.
if [ -z "$II_SYSTEM" -o ! -d "$II_SYSTEM/ingres" ] ;
then
echo 'II_SYSTEM must be defined for Ingres.'
exit 1
fi
sql iidbdb </dev/null >/dev/null 2>&1
if [ $? != 0 ] ; then
echo 'Ingres is not running, or the path is not set
up properly.'
echo 'Please make sure that Ingres has been started.'
exit 1
fi
dba=`infodb $dbName | sed -n -e '/Database
*:/s/^.*(.*,\([^)]*\)).*$/\1/p'`
# User has to be tmsdba.
userName=`IFS="()";set - \`id\`;echo $2`
if [ "$userName" != "$dba" ] ; then
echo "You are not the dba $dba for database $dbName"
exit 1
fi
# Decide which awk to use
AWK=awk
mach=`uname -s`
if [ "$mach" = 'SunOS' ] ; then
AWK=nawk
fi
# Dump optimizer statistics
statdump -zdl $dbName
# Generate list of all views, rules, procedures, and
dbevents by owner.
# Drop them all, they will be reapplied eventually
(after the upgrade).
# Note that by just dumping out all the view names,
we may get errors if a base view is dropped before a
dependent view. It seems a lot easier to just accept
that and tell the user to ignore any drop errors. The
```

```
alternative is to grind around in iidbdepends to
detect and deal with dependent views. Ugh.
While we're at it we might as well pick up the names
of tables with uncompressed HEAP storage structure.
In Ingres 6.4, unloaddb does not output any MODIFY
command for such tables. In order to touch all tables
(to ensure their goodness), we'll find uncompressed
heaps and arrange to re-heap them while we drop other
objects for that owner.
sql $dbName <<!EOF!
\\script /tmp/stuff.$dbName.$$
SELECT table_owner,'VIEW',table_name FROM iitables
WHERE table_type='V' AND table_owner <> '\$ingres'
UNION ALL SELECT dbp_owner,'PROCEDURE',dbp_name FROM
iiprocedure
WHERE dbp_owner<>'\$ingres'
UNION ALL SELECT rule_owner,'RULE',rule_name FROM
iirule
WHERE rule_owner<>'\$ingres'
UNION ALL SELECT event_owner,'DBEVENT',event_name
FROM iievent
WHERE event_owner<>'\$ingres'
UNION ALL SELECT table_owner,'HEAP',table_name FROM
iitables
WHERE table_type='T' AND table_owner<>'\$ingres'
AND storage_structure = 'HEAP' AND is_compressed =
'N'
ORDER BY 1,2,3;
COMMIT;
\\go
\\script
\\quit
!EOF!
if [ $? != 0 ] ; then
echo
echo "SQL returned error while processing database
$dbName"
exit 2
fi
# Transform output into "owner WHAT object-name"
```

```
/bin/ed /tmp/stuff.$dbName.$$ <<'!EOF!'
1,/^+---/d
1,/^+---/d
/^+---/,$d
1,$s/|//
1,$s/ *|/ /
1,$s/ *|/ /
1,$s/|//
w
q
!EOF!
# Generate awk script to take the stuff just dumped
and make DROP commands:
cat - >/tmp/awk$$ <<'!XX!'
BEGIN {curOwner=""}
{
      if ($1 != curOwner) {
        if (curOwner != "") {
            print "\\quit";
            print "!EOF!";
            print "if [ \$? != 0 ] ; then";
            print " echo 'Error running cleanup SQL'";
            print " exit 1";
            print "fi"
        }
        curOwner = $1;
        print "sql -u" curOwner,dbName,"<<'!EOF!'";
      }
      if ($2 != "HEAP") {
        print "DROP",$2,$3,";COMMIT;\\p\\g";
      } else {
        print "MODIFY",$3,"TO HEAP;COMMIT;\\p\\g";
      }
}
END {
      if (curOwner != "") {
        print "\\quit";
```

```
        print "!EOF!";
      }
}
!XX!
```
This awk call is known to be notwork on SunOS 4.x, but should be OK pretty much everywhere else. On SunOS 4.x you have to juggle the supplied variable (dbName) to a different spot in the command line.
```
$AWK -v "dbName=$dbName" -f /tmp/awk$$
/tmp/stuff.$dbName.$$
>/tmp/drop.$dbName.$$
if [ $? != 0 ] ; then
  echo 'awk error, perhaps your awk is strange?'
  exit 3
fi
# Ok, drop all sorts of stuff, re-heap heaps:
if [ -s /tmp/drop.$dbName.$$ ] ; then
  sh /tmp/drop.$dbName.$$
  if [ $? != 0 ] ; then
      echo "Error dropping objects from $dbName,
review the output log"
      exit 3
  fi
fi
```
Reapply storage structures, and incidentally drop indexes. Upgradedb doesn't have any problems with indexes, but you really ought to re-modify after the upgrade anyway, so save the index creates until then (Ingres II makes them more quickly anyway!)
```
for i in *_modify.sql ; do
  # Guard against no *_modify.sql files at all
  if [ "$i" != '*_modify.sql' ] ; then
      # Guard against empty files
      if [ -s "$i" ] ; then
        theUser=`expr "$i" : '\(.*\)_modify\.sql'`
        sql "-u$theUser" $dbName <$i
        if [ $? != 0 ] ; then
           echo "Error remodifying $dbName ($i)"
           exit 3
```

```
       fi
     fi
  fi
done
sysmod $dbName
if [ $? != 0 ] ; then
  echo
  echo "Sysmod error, perhaps database $dbName is
still in use."
  echo 'Make sure ALL users are locked out. Shut down
and restart'
  echo 'Ingres if necessary to ensure this. Then, try
again.'
  exit 3
fi
Do verifydb and infodb.
verifydb -mrun -sdbname $dbName -opurge
verifydb -mrun -sdbname $dbName -odbms_check
infodb $dbName | grep VALID
if [ $? = 0 ] ; then
  echo
  echo "Database $dbName seems OK, ready for upgrade
if verifydb output
was OK."
  echo
else
  echo
  echo "Database $dbName does not appear to be
consistent."
  exit 4
fi
rm -f /tmp/stuff.$dbName.$$ /tmp/drop.$dbName.$$
/tmp/awk$$
---------------- end of oi_prep.sh ----------------
-
```

Reserved Words

The following provides a complete table listing of Ingres II key words and indicates the contexts in which they are reserved. This list enables you to avoid assigning object names that conflict with reserved words.

NOTE: *THE KEY WORDS IN THIS LIST DO NOT NECESSARILY CORRESPOND TO SUPPORTED INGRES FEATURES. SOME WORDS ARE RESERVED FOR FUTURE OR INTERNAL USE, AND SOME WORDS ARE RESERVED TO PROVIDE BACKWARD COMPATIBILITY WITH OLDER FEATURES.*

In the table that follows, the column headings have the following meanings:

NON 6.4	These keywords were not included in Ingres 6.4 keyword reserved lists.
ISQL	Interactive SQL. These keywords are reserved by the DBMS.
ESQL	Embedded SQL. These keywords are reserved by the SQL preprocessors.
IQUEL	Interactive QUEL. These keywords are reserved by the DBMS.
EQUEL	Embedded QUEL. These keywords are reserved by the QUEL preprocessors.
4GL	These keywords are reserved in the context of SQL or QUEL in 4GL routines.

Note: The ESQL and EQUEL preprocessors also reserve forms statements.

	NON	SQL			QUEL		
Reserved in:	**6.4**	**ISQL**	**ESQL**	**4GL**	**IQUEL**	**EQUEL**	**4GL**
abort		*	*	*	*	*	*
activate			*			*	
add		*	*	*			
addform			*			*	
after	*			*			*
all		*	*	*	*	*	
alter		*		*			

	NON	SQL			QUEL		
Reserved in:	**6.4**	**ISQL**	**ESQL**	**4GL**	**IQUEL**	**EQUEL**	**4GL**
and		*	*	*	*	*	
any		*	*	*	*	*	
append					*	*	*
array	*			*			
as		*	*	*	*	*	*
asc		*		*			
at		*	*	*	*	*	*
authorization		*	*				
avg		*	*	*	*	*	
avgu			*		*	*	
before				*			*
begin		*	*	*	*		*
bell	*		*	*			
between		*	*	*			
breakdisplay			*			*	
by		*	*	*	*	*	*
byref	*	*	*	*			*
call			*	*		*	*
callframe	*			*			*
callproc	*	*		*			*
cascade	*	*	*				
check		*	*	*			
clear			*	*		*	*
clearrow			*	*		*	*
close		*	*		*		
column		*	*			*	
command			*			*	
comment	*			*			
commit		*	*	*			

Reserved in:	NON 6.4	SQL ISQL	ESQL	4GL	QUEL IQUEL	EQUEL	4GL
committed	*	*	*				
connect			*	*			
constraint	*	*	*	*			
constraints	*	*					
continue		*	*				
copy		*	*	*	*	*	*
count		*	*	*	*	*	
countu			*		*	*	
create		*	*	*	*	*	*
current		*	*	*			
current_user	*	*	*				
cursor		*	*				
datahandler	*		*				
dbms_password	*		*	*			
declare		*	*	*			*
default	*	*	*	*			*
define	*	*			*		*
delete	*	*	*	*	*	*	*
deleterow			*	*		*	*
desc				*			
describe	*	*	*				
descriptor			*				
destroy					*	*	*
direct	*			*			*
disable	*	*		*			
disconnect			*	*			
display			*	*		*	*
distinct		*	*	*			
distribute	*				*		

	NON	SQL			QUEL		
Reserved in:	**6.4**	**ISQL**	**ESQL**	**4GL**	**IQUEL**	**EQUEL**	**4GL**
do		*		*			*
down			*			*	
drop		*	*	*			
else		*		*			*
elseif		*		*			*
enable	*	*		*			
end		*	*	*	*	*	*
end-exec	*		*				
enddata			*			*	
enddisplay			*			*	
endforms			*			*	
endif		*		*			*
endloop		*	*	*		*	*
endretrieve						*	
endselect			*				
endwhile		*		*			*
escape		*	*	*			
exclude	*				*		
excluding	*	*	*		*		
execute		*	*	*	*		
exists		*	*	*			
exit				*		*	*
fetch		*	*				
field			*	*		*	
finalize			*			*	
for		*	*	*	*	*	
foreign	*	*	*	*			
formdata			*			*	
forminit			*			*	

Reserved in:	NON 6.4	SQL ISQL	ESQL	4GL	QUEL IQUEL	EQUEL	4GL
forms			*			*	
from		*	*	*	*	*	*
full	*	*	*	*			
get	*			*			
getform			*			*	
getoper			*			*	
getrow			*			*	
global	*	*	*	*			
goto			*				
grant		*	*	*			
granted	*	*	*	*			
group		*	*	*			
having		*	*	*			
help			*		*	*	
help_forms	*			*			*
help_frs			*			*	
helpfile			*	*		*	*
identified			*	*			
if		*	*	*			*
iimessage	*		*			*	
iiprintf	*		*			*	
iiprompt	*		*			*	
iistatement	*					*	
immediate		*	*	*			*
import	*	*					
in		*	*	*	*	*	
include			*		*		
index		*	*	*	*	*	*
indicator			*				

	NON	SQL			QUEL		
Reserved in:	**6.4**	**ISQL**	**ESQL**	**4GL**	**IQUEL**	**EQUEL**	**4GL**
ingres						*	
initial_user	*	*	*				
initialize			*	*		*	*
inittable			*	*		*	*
inner	*	*	*	*			
inquire_4gl	*			*			*
inquire_equel						*	
inquire_forms	*			*			*
inquire_frs			*			*	
inquire_ingres	*		*	*		*	*
inquire_sql			*	*			
insert		*	*	*			
insertrow			*	*		*	*
integrity		*	*		*		*
into		*	*	*	*	*	*
is		*	*	*	*	*	*
isolation	*	*					
join	*	*	*	*			
key	*	*	*	*			*
left	*	*	*	*			
level	*	*	*		*	*	
like		*	*	*			
loadtable			*	*		*	*
local	*	*					
max		*	*	*	*	*	
menuitem			*			*	
message		*	*	*		*	*
min		*	*	*	*	*	
mode	*			*			*

Reserved in:	NON 6.4	SQL ISQL	ESQL	4GL	QUEL IQUEL	EQUEL	4GL
modify		*	*	*	*	*	*
module	*	*					
move	*				*		
natural	*	*	*				
next			*	*		*	
noecho	*			*			*
not		*	*	*	*	*	
notrim			*			*	
null		*	*	*		*	*
of		*	*	*	*	*	*
off	*			*			*
on		*	*	*	*	*	*
only	*				*		*
open		*	*		*		
option		*					
or		*	*	*	*	*	
order		*	*	*	*	*	*
out			*			*	
outer	*	*					
param						*	
permit		*	*		*		*
prepare		*	*				
preserve	*	*	*				
primary	*	*	*	*			
print			*		*	*	
printscreen			*	*		*	*
privileges		*					
procedure		*	*	*			*
prompt			*	*		*	*

Reserved in:	NON 6.4	SQL ISQL	ESQL	4GL	QUEL IQUEL	EQUEL	4GL
public		*	*				
purgetable	*		*	*			*
putform			*			*	
putoper			*			*	
putrow			*			*	
qualification	*			*			*
raise	*	*		*			
range					*	*	*
read	*		*		*		
redisplay			*	*		*	*
references	*	*	*	*			
referencing		*		*			
register		*	*	*	*	*	*
relocate		*	*	*	*	*	*
remove		*	*	*		*	*
rename	*				*		
repeat		*	*	*		*	*
repeatable	*	*	*				
repeated			*	*			
replace					*	*	*
replicate	*				*		
restrict	*	*	*				
resume			*	*		*	*
retrieve					*	*	*
return		*		*			*
revoke		*	*	*			
right	*	*	*	*			
role	*	*	*	*			
rollback		*	*	*			

	NON	SQL			QUEL		
Reserved in:	**6.4**	**ISQL**	**ESQL**	**4GL**	**IQUEL**	**EQUEL**	**4GL**
rows	*	*	*				
run	*			*			*
save		*	*	*	*	*	*
savepoint		*	*	*	*	*	*
schema	*	*	*				
screen			*	*		*	*
scroll			*	*		*	*
scrolldown			*			*	
scrollup			*			*	
section			*				
select		*	*	*			
serializable	*	*					
session	*	*	*	*			
session_user	*	*	*				
set		*	*	*	*	*	*
set_4gl	*			*			*
set_equel						*	
set_forms	*			*			*
set_frs			*			*	
set_ingres	*		*	*		*	*
set_sql			*	*			
sleep			*	*		*	*
some		*	*	*			
sort					*	*	*
sql		*					
stop			*				
submenu			*			*	
sum		*	*	*	*	*	
sumu			*		*	*	

	NON	SQL			QUEL		
Reserved in:	6.4	ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
system	*			*			*
system_user	*		*				
table		*	*	*			
tabledata			*			*	
temporary	*	*	*				
then		*	*	*			*
to		*	*	*	*	*	*
type	*			*			
uncommitted	*	*	*				
union		*	*	*			
unique		*	*	*	*	*	*
unloadtable			*	*		*	*
until		*	*	*	*	*	*
up			*			*	
update		*	*	*	*		
user		*	*	*			
using		*	*				
validate			*	*		*	*
validrow			*	*		*	*
values		*	*	*			
view		*	*		*		*
when	*	*	*				
whenever			*				
where		*	*	*	*	*	*
while		*		*			*
with		*	*	*	*	*	*
work		*		*			
write	*				*		

Double Keyword	NON	SQL			QUEL		
Reserved in:	6.4	ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
add privileges	*			*			
after field	*			*			*
alter default	*		*	*			
alter group		*	*	*			
alter location	*	*	*	*			
alter profile	*	*	*	*			
alter role		*	*	*			
alter security_audit	*	*	*	*			
alter table	*	*	*	*			
alter user	*	*	*	*			
array of	*			*			
before field	*			*			*
begin declare	*		*				
begin exclude	*		*				
begin transaction		*	*	*	*	*	*
by group	*			*			
by role	*	*		*			
by user	*	*		*			
call on	*			*			
call procedure	*			*			
class of	*			*			
clear array	*		*				
close cursor			*		*	*	
comment on	*	*	*	*			
connect to	*			*			
copy table	*			*			
create dbevent		*	*	*			
create domain	*		*				
create group		*		*			

Double Keyword	NON	SQL			QUEL		
Reserved in:	6.4	ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
create integrity	*	*		*			
create link		*	*				
create location	*	*	*	*			
create permit	*	*		*			
create procedure	*			*			
create profile	*	*	*	*			
create role		*	*	*			
create rule		*	*	*			
create security_alarm	*	*	*	*			
create synonym	*	*	*	*			
create user	*	*	*	*			
create view	*	*		*			
current installation	*			*			
define cursor					*		
declare cursor						*	
define integrity					*	*	*
define link						*	
define location					*		
define permit					*	*	*
define qry		*			*		*
define query		*			*		
define view					*	*	*
delete cursor					*	*	
describe form	*		*				
destroy integrity					*	*	*
destroy link						*	
destroy permit					*	*	*
destroy table						*	
destroy view	*						*

Double Keyword	NON	SQL			QUEL		
Reserved in:	6.4	ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
direct connect			*	*		*	*
direct disconnect			*	*		*	*
direct execute			*	*			*
disable security_audit	*	*	*	*			
disconnect current	*			*			
display submenu	*			*			*
drop dbevent		*	*	*			
drop domain	*		*				
drop group		*		*			
drop integrity	*	*		*			
drop link		*	*	*			
drop location	*	*	*	*			
drop permit	*	*		*			
drop privileges	*			*			
drop procedure	*			*			
drop profile	*	*	*	*			
drop role		*	*	*			
drop rule		*	*	*			
drop security_alarm	*	*	*	*			
drop synonym	*	*	*	*			
drop user	*	*	*	*			
drop view	*	*		*			
each row	*		*				
each statement	*		*				
enable security_audit	*	*	*	*			
end exclude	*		*				
end transaction		*	*	*	*	*	*
exec sql	*		*				
execute immediate	*			*			

Double Keyword	NON	SQL			QUEL		
Reserved in:	6.4	ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
execute on	*			*			
execute procedure	*			*			
foreign key	*			*			
for deferred		*			*		
for direct		*			*		
for readonly		*			*		
for retrieve	*				*		
for update					*		
from group		*		*			
from role		*		*			
from user		*		*			
full join	*	*		*			
full outer	*	*		*			
get attribute	*		*				
get data	*		*				
get dbevent	*		*	*			
get global	*		*				
global temporary	*			*			
help all	*		*				
help comment	*	*					
help integrity			*			*	
help permit			*			*	
help table	*	*					
help view			*			*	
identified by	*			*			
inner join	*	*		*			
is null					*		
isolation level	*		*		*		
left join	*	*		*			

Double Keyword	NON	SQL			QUEL		
Reserved in:	6.4	ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
left outer	*	*		*			
modify table	*			*			
not like	*	*		*			*
not null	*				*		
on commit	*	*	*	*			
on current	*	*					
on database		*		*			
on dbevent		*		*			*
on location	*	*		*			
on procedure	*	*					
only where					*		
open cursor			*		*	*	
order by					*		
primary key	*			*			
procedure returning	*			*			*
put data	*		*				
raise dbevent		*	*	*			
raise error		*					
read only	*		*				
read write	*		*				
register dbevent		*	*	*			
register table	*						*
register view	*			*			*
remote system_password	*		*				
remote system_user	*		*				
remove dbevent		*	*	*			
remove table	*						*
remove view	*			*			*

Double Keyword	NON	SQL			QUEL		
Reserved in:	6.4	ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
replace cursor			*		*	*	*
resume entry	*			*			*
resume menu	*			*			*
resume next	*			*			*
resume nextfield	*			*			*
resume previousfield	*			*			*
retrieve cursor			*		*	*	
right join	*	*		*			
right outer	*	*		*			
run submenu	*			*			*
send userevent	*			*			
session group	*			*			
session role	*			*			
session user	*			*			
set aggregate	*	*			*		
set attribute	*		*				
set autocommit	*	*			*		
set cache	*	*			*		
set connection	*		*	*			*
set cpufactor	*	*			*		
set date_format	*	*			*		
set ddl_concurrency	*	*					
set deadlock	*	*			*		
set decimal	*	*			*		
set flatten	*	*			*		
set global	*		*				
set io_trace	*	*			*		
set j_freesz1	*	*			*		
set j_freesz2	*	*			*		

Double Keyword	NON	SQL			QUEL		
Reserved in:	6.4	ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
set j_freesz3	*	*			*		
set j_freesz4	*	*			*		
set j_sortbufsz	*	*			*		
set jcpufactor	*				*		
set joinop	*	*			*		
set journaling	*	*			*		
set lock_trace	*	*			*		
set lockmode	*	*			*		
set logdbevents	*	*					
set log_trace	*	*			*		
set logging	*	*			*		
set maxconnect	*	*			*		
set maxcost	*	*			*		
set maxcpu	*	*			*		
set maxidle		*			*		
set maxio		*			*		
set maxpage	*	*			*		
set maxquery	*	*			*		
set maxrow		*			*		
set money_format	*	*			*		
set money_prec	*	*			*		
set nodeadlock	*	*			*		
set noflatten	*	*			*		
set noio_trace	*	*			*		
set nojoinop	*	*			*		
set nojournaling	*	*			*		
set nolock_trace	*	*			*		
set nologdbevents	*	*					
set nolog_trace	*	*			*		

Double Keyword	NON	SQL			QUEL		
Reserved in:	6.4	ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
set nologging	*	*			*		
set nomaxconnect	*	*			*		
set nomaxcost	*	*			*		
set nomaxcpu	*	*			*		
set nomaxidle	*	*			*		
set nomaxio	*	*			*		
set nomaxpage	*	*			*		
set nomaxquery	*	*			*		
set nomaxrow	*	*			*		
set nooptimizeonly	*	*			*		
set noprintdbevents	*	*					
set noprintqry	*	*			*		
set noprintrules	*	*					
set noqep	*	*			*		
set norules	*	*					
set nosql	*				*		
set nostatistics	*	*			*		
set notrace	*	*			*		
set optimizeonly	*	*			*		
set printdbevents	*	*					
set printqry	*	*			*		
set qbufsize	*	*			*		
set qep	*	*			*		
set query_size	*	*			*		
set random_seed	*	*			*		
set result_structure	*	*			*		
set ret_into	*	*			*		
set role	*		*				
setrow deleted	*	*					

Double Keyword	NON	SQL			QUEL		
Reserved in:	6.4	ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
set rowlabel_visible	*		*				
set rules		*					
set session	*	*			*		
set sortbufsize	*	*			*		
set sql	*				*		
set statistics	*	*			*		
set trace	*	*			*		
set transaction	*		*				
set update_rowcount	*	*			*		
set work	*	*					
system user	*			*			
to group		*		*			
to role		*		*			
to user		*	*	*			
user authorization	*			*			
with null					*		
with short_remark	*	*					