



Migration Guide

r3



Computer Associates®

A00193-1E

This documentation and related computer software program (hereinafter referred to as the "Documentation") is for the end user's informational purposes only and is subject to change or withdrawal by Computer Associates International, Inc. ("CA") at any time.

This documentation may not be copied, transferred, reproduced, disclosed or duplicated, in whole or in part, without the prior written consent of CA. This documentation is proprietary information of CA and protected by the copyright laws of the United States and international treaties.

Notwithstanding the foregoing, licensed users may print a reasonable number of copies of this documentation for their own internal use, provided that all CA copyright notices and legends are affixed to each reproduced copy. Only authorized employees, consultants, or agents of the user who are bound by the confidentiality provisions of the license for the software are permitted to have access to such copies.

This right to print copies is limited to the period during which the license for the product remains in full force and effect. Should the license terminate for any reason, it shall be the user's responsibility to return to CA the reproduced copies or to certify to CA that same have been destroyed.

To the extent permitted by applicable law, CA provides this documentation "as is" without warranty of any kind, including without limitation, any implied warranties of merchantability, fitness for a particular purpose or noninfringement. In no event will CA be liable to the end user or any third party for any loss or damage, direct or indirect, from the use of this documentation, including without limitation, lost profits, business interruption, goodwill, or lost data, even if CA is expressly advised of such loss or damage.

The use of any product referenced in this documentation and this documentation is governed by the end user's applicable license agreement.

The manufacturer of this documentation is Computer Associates International, Inc.

Provided with "Restricted Rights" as set forth in 48 C.F.R. Section 12.212, 48 C.F.R. Sections 52.227-19(c)(1) and (2) or DFARS Section 252.227-7013(c)(1)(ii) or applicable successor provisions.

© 2004 Computer Associates International, Inc.

All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

Contents

Chapter 1: Introduction

Planning the Upgrade	1-1
New Ingres Features	1-1
Upgrade Types	1-2
Upgradedb Method	1-2
Unload/Reload Method	1-2
Ingres Releases	1-3
From Releases Prior to Ingres 6.4	1-3
From Ingres 6.4	1-3
From Releases Newer than Ingres 6.4	1-4
From a 32-bit to a 64-bit Release	1-4
To Member-Aligned Alpha OpenVMS (axm.vms)	1-5
Binary Level Support	1-5
Installations and Hardware	1-5
Possible Hardware Setups	1-6
Overview of Upgrade Procedure	1-6
Application Issues	1-7
The Test Plan for Applications	1-8
Keys to Success	1-8
Debugging	1-9

Chapter 2: Getting Started

Note on Platform-specific Examples	2-1
Upgrading from Ingres 6.4	2-1
Creating a New Ingres Development Installation	2-2
Preparing Your Applications	2-3
New Reserved Keywords	2-3
Report-Writer Syntax Change	2-4
Report-Writer Runtime Parameter Errors	2-4
Loading Databases and Applications into the New Installation	2-5

Creating Users	2-5
Moving Databases	2-5
Moving Catalogs	2-6
Moving Distributed Option Databases	2-6
The system_maintained Column Name	2-6
Compiling Applications	2-7
Testing	2-7
Preparing Your System	2-8
System Monitoring Shellscrips	2-8
Checkpoint Template Changes	2-8
Other Checkpoint and Rollforward Changes	2-9
Backup and Restore	2-9
Shared Library Search Path	2-9
UNIX Kernel Parameters	2-10
Testing	2-11
Performance Testing	2-11
System Administrator Procedures	2-11
Practicing the Upgrade	2-11
Before the Live Upgrade	2-12
OpenROAD 4.0 Image File Formats	2-12

Chapter 3: Upgrading Using Upgradedb

Upgradedb Upgrade Procedure	3-1
Step 1: Disable User Access	3-1
Step 2: Disable Remote Command Server	3-2
Step 3: Shut Down Ingres and Back Up System	3-2
Step 4: [Each DB Including iidbdb] Clean the Database	3-3
Step 5: [Each DB] Record Database Information	3-3
Step 6: [Each DB Including iidbdb] Checkpoint and Turn Off Journaling	3-3
Step 7: Shut Down Ingres	3-4
Step 8: Preserve Site Modifications	3-4
Visual DBA Configurations	3-5
Step 9: Delete Install Directory	3-5
Step 10: Install Ingres	3-5
Upgrading to Older Versions That Require a Patch	3-6
Step 11: Create imadb Database	3-6
Step 12: Restore Site Modifications	3-7
Step 13: Start Ingres	3-7
Step 14: Run Upgradedb Utility	3-7
Step 15: Review Ingres Configuration	3-8

Step 16: [Each DB] Reapply Optimizer Statistics (Optional)	3-8
Step 17: [Each DB including iidbdb] Checkpoint the Database	3-8
Step 18: Install Upgraded Applications	3-8

Chapter 4: Upgrading Using Unload/Reload

Two Variations	4-1
Unload/Reload Upgrade Procedure	4-2
Step 1: [Each DB Including iidbdb] Create Unload Directories	4-2
Step 2: [Each DB] Run Unloadddb	4-2
Step 3: [Each DB] Check for Obsolete Users	4-3
Step 4: [Each DB Including iidbdb] Checkpoint the Database (Optional)	4-3
Step 5: Disable User Access	4-3
Step 6: Disable Remote Command Server	4-4
Step 7: Shut Down Ingres and Back Up System	4-4
Step 8: [Each DB] Unload the Database	4-5
Step 9: [Each DB] Print Optimizer Statistics (Optional)	4-5
Step 10: [Each DB] Record Database Information	4-5
Step 11: Record Database Privileges	4-6
Step 12: Save Users, Groups, and Roles	4-6
Step 13: [Each DB] Destroy the Database	4-7
Step 14: Clean iidbdb Database	4-7
Step 15: Shut Down Ingres	4-7
Step 16: Disable Ingres Startup	4-7
Step 17: Preserve Site Modifications	4-8
Visual DBA Configurations	4-9
Step 18: Delete Install Directory	4-9
Step 19: Install Ingres	4-9
Upgrading to Older Versions That Require a Patch	4-10
Step 20: Create imadb Database	4-10
Step 21: Restore Site Modifications	4-11
Step 22: Review Ingres Configuration	4-11
Step 23: Set Up Ingres Net	4-11
Step 24: Start Ingres	4-12
Step 25: Recreate Users, Groups, and Roles	4-12
Step 26: Recreate Locations	4-12
Step 27: [Each DB] Recreate the Database	4-13
Step 28: [Each DB] Extend the Database	4-13
Step 29: Recreate Database Privileges	4-13
Step 30: [Each DB] Fix FE Reload Script	4-14
Step 31: [Each DB] Reload the Database	4-15

Step 32: [Each DB] Upgrade Front-End Catalogs	4-15
Step 33: [Each DB] Reapply Optimizer Statistics	4-15
Step 34: [Each DB including iidbdb] Checkpoint the Database.....	4-15
Step 35: Install Upgraded Applications	4-16

Appendix A: Considerations for Alpha OpenVMS

OpenVMS Requirements	A-1
Installing Ingres	A-1
Mounting the CD	A-1
Running VMSINSTAL	A-2
Known Installation Issues	A-2
Schema Checking	A-3
Rebuilding Applications	A-3
Building Member_Aligned Against Ingres 2.6 or r3	A-3
For C Applications	A-4
For COBOL Applications	A-5

Appendix B: Upgrading from Ingres 6.4

Considerations for Ingres 6.4	B-1
Preparing Your Applications	B-1
UPDATE ... FROM Semantics Change	B-1
Decimal Constant Semantics Change	B-2
Greater Sensitivity to BYREF Errors	B-2
Journaling On by Default	B-3
Greater Sensitivity to Arithmetic Errors	B-3
4GL TABLE_KEY Type Conversions.....	B-3
User-Defined Data Type Changes	B-3
Summary	B-4
Preparing Your System.....	B-4
Ingres Startup and Shutdown	B-4
ingprenv Replaces ingprenv1	B-4
Archiver Exit Shellscript.....	B-4
Transaction Log Size	B-5
Upgrading from 6.4 Using Upgradedb	B-5
Procedure.....	B-6
Step 1: [Each DB Including Distributed Option DDBs] Create Unload Directory	B-6
Step 2: [Each DB Including Distributed Option DDBs] Run Unloaddb	B-6
Step 3: [Each DB Including Distributed Option DDBs] Check for Obsolete Users	B-7

Step 4: [Each DB] Edit the Unloaddb Output	B-7
Step 5: [Each DB Including iidbdb] Checkpoint the Database (Optional)	B-8
Step 6: Disable User Access	B-8
Step 7: Shut Down Ingres and Back Up System	B-8
Step 8: [Each DB] Print Optimizer Statistics (Optional)	B-9
Step 9: [Each DB] Remove Non-table Objects	B-9
Step 10: [Each DB] Record Database Information	B-10
Step 11: Clean iidbdb Database	B-11
Step 12: [Each DB Including iidbdb] Checkpoint and Turn Off Journaling	B-11
Step 13: Record Ingres Configuration	B-11
Step 14: Shut Down Ingres	B-12
Step 15: Disable Ingres Startup	B-12
Step 16: Preserve Site Modifications	B-12
Step 17: Fix Logins	B-13
Step 18: Save Ingres Settings	B-13
Step 19: Clean Up Ingres 6.4	B-14
Step 20: Create Work Location	B-14
Step 21: Install Ingres	B-15
Upgrading to Versions That Require a Patch	B-15
Step 22: Create imadb Database	B-15
Step 23: Restore Site Modifications	B-16
Step 24: Start Ingres	B-16
Step 25: Run Upgradedb Utility	B-16
Step 26: Configure Ingres	B-17
Step 27: Set Up Ingres Net	B-17
Step 28: [Each DB] Recreate Objects	B-18
Step 29: [Each DB] Reapply Storage Structures	B-18
Step 30: [Each DB] Reapply Optimizer Statistics	B-18
Step 31: [Each DB including iidbdb] Checkpoint the Database	B-19
Step 32: Install Upgraded Applications	B-19
Upgrading from 6.4 Using Unload/Reload	B-19
Two Upgrade Types	B-20
Front-end Catalogs	B-20
Procedure	B-20
Step 1: [Each DB Including iidbdb] Create Unload Directory	B-21
Step 2: [Each DB] Run Unloaddb	B-21
Step 3: [Each DB] Check for Obsolete Users	B-21
Step 4: [Each DB Including iidbdb] Checkpoint the Database (Optional)	B-22
Step 5: Disable User Access	B-22
Step 6: Shut Down Ingres and Back Up System	B-22
Step 7: [Each DB] Unload the Database	B-23

Step 8: [Each DB] Print Optimizer Statistics (Optional)	B-23
Step 9: [Each DB] Record Database Information	B-23
Step 10: Record Database Privileges	B-24
Step 11: Save Users, Groups, and Roles	B-24
Step 12: [Each DB] Destroy the Database	B-25
Step 13: Clean iidbdb Database	B-25
Step 14: Record Ingres Configuration	B-25
Step 15: Shut Down Ingres	B-26
Step 16: Disable Ingres Startup	B-26
Step 17: Preserve Site Modifications	B-26
Step 18: Fix Logins	B-27
Step 19: Save Ingres Settings	B-28
Step 20: Clean Up Ingres 6.4	B-28
Step 21: Create Work Location	B-28
Step 22: Install Ingres	B-29
Upgrading to Versions That Require a Patch	B-29
Step 23: Create imadb Database	B-29
Step 24: Restore Site Modifications	B-30
Step 25: Configure Ingres	B-30
Step 26: Set Up Ingres Net	B-31
Step 27: Start Ingres	B-31
Step 28: Recreate Users, Groups, and Roles	B-31
Step 29: Recreate Locations	B-32
Step 30: [Each DB] Recreate the Database	B-32
Step 31: [Each DB] Extend the Database	B-33
Step 32: Recreate Database Privileges	B-33
Step 33: [Each DB] Fix FE Reload Script	B-34
Step 34: [Each DB] Reload the Database	B-34
Step 35: [Each DB] Upgrade Front-end Catalogs	B-34
Step 36: [Each DB] Reapply Optimizer Statistics	B-35
Step 37: [Each DB including iidbdb] Checkpoint the Database	B-35
Step 38: Install Upgraded Applications	B-35
Corresponding Parameter Names	B-35
Parameters in 6.4 rundbms.opt File	B-36
Locking and Logging System Parameters	B-38

Appendix C: Troubleshooting Upgradedb Problems

Troubleshooting Tips	B-1
----------------------------	-----

Appendix D: Keywords

Table Key	D-1
Reserved Single Keywords	D-2
Reserved Double Keywords	D-12
Other Reserved Keywords	D-22

Appendix E: Features Introduced in Advantage Ingres 2.6

User-Visible Language Enhancements	E-1
Row Producing Procedures	E-1
SUBSTRING Function	E-2
New Aggregate Functions	E-2
Increased Maximum Size of Character Data Types	E-2
User-Visible DBA Enhancements	E-3
Usermod Utility	E-3
Auditdb Utility	E-3
Copydb Utility	E-3
Raw Location Support	E-4
GatherWrite Threads	E-4
XML Import/Export Utility	E-4
Journal Analyzer	E-4
Import Assistant	E-4
Automated Creation of Location Directories	E-5
Remote Command Server Enhancements	E-6
Microsoft Transaction Server Support	E-6
Concurrent Rollback	E-6
Internal Performance Enhancements	E-6
Aggregate Sort Nodes	E-6
Composite Histograms	E-7
Optimizer Support for Hash Joins	E-7
Locking System Performance Improvements	E-7
Preallocated RSB/LKBs	E-7
Miscellaneous Locking System Improvements	E-7
Logging System Performance Improvements	E-8
Buffer Manager Performance Improvements	E-8
Operating System Integration	E-8
64-Bit Operating Systems	E-8
Operating System Thread Implementation on Linux	E-9
Ingres ICE Enhancements	E-9
Development Environment	E-9

ODBC Enhancements	E-9
Supported Functions	E-9
Unavailable Features	E-10
JDBC Enhancements	E-10
Support for Unicode	E-10
New Character Sets to Support Euro Currency Symbol	E-11

Appendix F: Features Introduced in Ingres II 2.5

Sort Enhancements	F-2
QEF Sort Enhancements	F-2
DMF Sort Enhancements	F-2
Parallel Sort Techniques	F-3
ANSI/ISO Constraint Enhancements	F-3
Large Cache Support	F-4
Dynamic Write Behind Threads	F-5
Partitioned Transaction Log File	F-5
Optimizer and Optimizedb Enhancements	F-6
Read-only Database Support	F-6
New SQL Functionality	F-7
Order By/Group By Expression	F-7
CASE Expression	F-8
Parallel Index Creation	F-8
SELECT Enhancement	F-8
Bit-wise Operator Support	F-9
Aggregate Functions	F-9
Miscellaneous Functions	F-9
Extended Date Support	F-10
Large File Support	F-10
Large Catalogs	F-10
Row Locking for System Catalogs	F-10
Update Mode Locking	F-11
Value Locking for Serializable Transaction with Equal Predicate	F-11
Query Optimization and Execution Enhancements	F-11
Ingres Star Features	F-11
Ingres Net Features	F-12
Ingres ICE Features	F-12
Security	F-12
Session Management	F-13
Storage Management	F-13
Macro Language Extensions	F-13

Visual DBA Features	F-14
Replicator Enhancements	F-14
Generic Replicator Server	F-14
Increased Replicator Concurrency	F-14
OpenAPI Enhancements	F-15

Appendix G: Features Introduced in Ingres II 2.0

Variable Page Size	G-2
New Page Format for Larger Page Size	G-2
Larger Tuple Support	G-2
Distributed Multi-Cache Management	G-3
Enhanced Performance of Locking/Logging and Buffer Manager	G-3
Interval Based Deadlock Detection	G-3
Row Level Locking and Transaction Isolation Levels	G-4
Alter Table Support	G-5
Async I/O Support	G-5
Parallel Backup and Restore	G-6
Parallel Checkpointing to Disk	G-6
Parallel Checkpointing to Tape	G-6
Parallel Rollforwarddb from Disk	G-6
Parallel Rollforwarddb from Tape	G-6
Fast Load Support	G-7
R-tree Support: A Spatial Index for Ingres II 2.0	G-7
Spatial Data Types and Operators	G-7
Statement Level Rules	G-8
Temporary Tables as Procedure Parameters	G-8
Other Optimizer Enhancements	G-9
Operating System Thread Support	G-9
Table Cache Priorities	G-10
Transaction Access Mode	G-10
Soundex Function	G-10
Ingres Star Features	G-10
Ingres Net Features	G-11
Protocol Bridge Support	G-11
Data-Stream Compression Support	G-11
SNA Duplex Support	G-12
DECNet/OSI Support	G-12
Ingres OpenAPI Enhancements	G-12
Multi-Threaded OpenAPI	G-12
OpenAPI Support for Autocommit	G-13

Enhanced OpenAPI Support for Database Events	G-13
Ingres ICE Features	G-13
Visual DBA Features	G-14
Server-based Replication	G-15

Index

Introduction

This guide, together with the other guides in the documentation set, will assist in the planning and execution of a successful upgrade of Ingres®. By taking advantage of the enhanced features and functions at the next level, you can obtain superior support services and reap the benefits associated with running the latest version of this product.

Planning the Upgrade

The most important part of any upgrade is to prepare a detailed plan. A detailed plan can prevent problems, which is the key to any upgrade. The plan should include items as simple as how long it will take to complete a backup and how to verify that the data is complete and secure.

The plan should then be tested, preferably with a copy of the production system data. Testing reveals areas that may cause problems during the upgrade of the production system.

You should then implement the plan, but **only** after preliminary testing is complete.

The best strategy for upgrading is to first implement any compatibility fixes in the current environment. When the databases and applications are ready, test them in that environment, practice the upgrade, and then perform the upgrade.

Do not use any new features until the upgrade is successfully implemented. Doing this keeps to a minimum the number of variables at each step.

New Ingres Features

Included as appendixes in this guide are new features for Advantage Ingres 2.6, Ingres II 2.5, and Ingres II 2.0. The information is provided so that after the upgrade is complete and running successfully for several days, you can begin using the new features.

Upgrade Types

There are two options for upgrading your production systems:

- The upgradedb utility
- The unload/reload method

You can mix the two upgrade types, upgrading some databases while reloading others.

Upgradedb Method

The upgradedb utility allows for a fast, in-place upgrade path for an older version Ingres database, with no additional disk space requirements. Because upgradedb is faster, it is typically the recommended way of upgrading.

Preparing for a safe and reliable upgradedb, however, can take time, especially when upgrading from Ingres 6.4. Older versions of upgradedb have some specific issues that must be accommodated.

Databases using the system-maintained logical key feature are best upgraded using upgradedb. Tables that contain SYSTEM_MAINTAINED table_key or object_key columns cannot be safely unloaded and reloaded without additional work. The reload step generates all new logical key values. If there are other tables referencing the logical key columns, the new values must somehow be manually propagated to those other tables.

Unload/Reload Method

A database unload/reload ensures a clean start with a fresh database. Depending on the kind of table data, additional disk space may be needed to do the unload and reload; this could be as large as three to five times the space of the database that is to be upgraded. For instance, compressed tables with wide CHAR or VARCHAR columns can expand substantially when unloaded.

The unload/reload process takes longer than upgradedb, thus increasing the downtime of the production system. However, it ensures a clean final installation.

A database that has been running for years, perhaps surviving a number of system crashes and hardware failures, may have suffered hidden damage that can confuse the upgradedb utility. 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. Such a database may have worktables owned by a user who no longer exists, or may be missing table data files. An unload/reload upgrade may be a better choice for this database.

The typical unload/reload upgrade uses the original Ingres installation as a base. The system databases iidbdb and imadb are upgraded in-place with `upgradedb`, even if user databases are unloaded/reloaded. A variation of the unload/reload method uses a brand new installation (perhaps even on a different machine). When this is done, additional work is needed to transfer iidbdb information (users, groups, roles, and database and installation privileges) to the new installation.

Ingres Releases

The release of Ingres that you are upgrading from affects the type of upgrade you choose.

From Releases Prior to Ingres 6.4

You must use an unload/reload upgrade if you are starting with a version of Ingres prior to release 6.4. Furthermore, you must install Ingres into a new, fresh installation; the original installation cannot be upgraded in-place.

From Ingres 6.4

You can use either method to upgrade from Ingres 6.4. Regardless of the method chosen, however, an upgrade from Ingres 6.4 requires more planning and preparation than upgrades from newer versions.

Significant preparation is needed to use `upgradedb` safely against a 6.4 database. The procedures set forth in this guide allow you to use the `upgradedb` method successfully.

On the other hand, since most 6.4 databases are several years old, you may choose to use an unload/reload upgrade. Keep in mind that an unload/reload upgrade takes substantially more time and resources.

From Releases Newer than Ingres 6.4

Upgradedb is the recommended type when upgrading from OpenIngres 1.2 or 2.0, Ingres II 2.0 or 2.5, or Advantage Ingres 2.6. The upgrade is internally much simpler than the upgrade from 6.4. In addition, there are fewer application-level incompatibilities among newer versions.

An unload/reload upgrade is possible, but is slower and requires more disk space than an upgradedb upgrade.

OpenIngres 1.2: If you are starting with OpenIngres 1.2, any tables having long varchar, long binary, or long spatial data must be unloaded under 1.x and reloaded into Advantage Ingres 2.6. The format of the blob extension tables has changed. The remainder of the database can be upgraded with upgradedb; however, it is probably simplest to use a full unload/reload upgrade with any databases containing “long” datatypes.

From a 32-bit to a 64-bit Release

You can upgrade your 32-bit Ingres database for use with 64-bit Ingres by running the upgradedb utility. The upgrade process that is specific to converting a database from the 32-bit to 64-bit architecture redefines views, rules, integrities, and QUEL permits. The data in user tables is **not** affected by the 32-bit to 64-bit upgrade.

The upgradedb program does the following:

- Redefines the standard catalog views (iitables, iicolumns, and so on)
- Generates an SQL script to drop and redefine views, rules, integrities, and QUEL permits
- Executes the SQL script

The generated SQL script can be found in
\$II_SYSTEM/ingres/files/upgradedb/DBNAME/DBNAME.i0.

The SQL output can be found in \$II_SYSTEM/ingres/files/upgradedb/DBA-NAME/DBNAME.o01.

If your database contains an object that cannot be redefined, the upgradedb may fail to redefine all objects. You can use the SQL script and output in \$II_SYSTEM/ingres/files/upgradedb to determine the point of failure. If necessary, contact technical support for assistance.

To Member-Aligned Alpha OpenVMS (axm.vms)

If you are using OpenVMS on Alpha hardware, and are upgrading to the member-aligned version of Ingres (axm.vms) from a non-member-aligned version (axp.vms), you must use unload/reload. Upgradedb is not available due to shifts in table data positions caused by the new alignment. For instructions, see the appendix “[Considerations for Alpha OpenVMS](#).”

Binary Level Support

Ingres 3 provides support for applications built against previous versions of Ingres, back to and including Ingres 6.4.

You can run applications built with any version of OpenIngres 1.x or Ingres II 2.x, accessing an Ingres 3 server, without rebuilding the application.

Applications built against Ingres 6.4 expect to access an older version of the Ingres message files. You can run applications built with Ingres 6.4 against an Ingres 3 server, as long as either the application is running across Ingres Net, or Ingres 3 was installed with the 6.4 message files. However, in all cases, we recommend that you rebuild all applications with Ingres 3.

Installations and Hardware

For a safe and orderly upgrade, at least four Ingres installations are needed:

- Original version production installation
- Original version development installation
- Installation for testing the upgrade
- New version development installation for preparing and testing applications

Four machines can be used, putting each installation on its own machine. More commonly, however, the two development installations share a machine. Since there is usually some traffic between these two installations during preparation, sharing a machine is convenient.

Note: If you are using Windows, you need a separate machine for each installation. Versions prior to Advantage Ingres 2.6 do not support multiple installations on one Windows machine.

If possible, keep the installations away from the production machine. If there is no hardware available for the above installations, you may temporarily need additional hardware.

Possible Hardware Setups

Three Machines	<p>The recommended minimum hardware setup is three machines:</p> <ul style="list-style-type: none">■ Development (both old and new versions)■ Test■ Production
Two Machines	<p>It is possible – but not recommended – to use two machines:</p> <ul style="list-style-type: none">■ Development (possibly including a test installation)■ Production <p>The latter configuration is not preferred, as the test installation shares a machine with development, so it will not mimic your production installation as closely. In addition, the more installations a machine has, the more chance for error.</p>
One Machine	<p>Using a single machine for an upgrade is possible, but not recommended. There is too great a likelihood of accidentally working in the wrong installation and damaging production.</p> <p>Note: There is no remote installation procedure for Ingres. The machine must have local media support (CD-ROM or tape); otherwise, you will have to copy the distribution files from wherever the CD-ROM or tape drive is situated.</p>

Overview of Upgrade Procedure

The following describes the overall strategy for the upgrade.

Tip: Back up all data before starting.

1. **Copy** – Duplicate the databases to be upgraded into the new version development installation, and make a copy of all associated applications. It is important that your original version development installation remain; if the upgrade is unpredictably delayed, you will still have your original environment in which to fix mission-critical applications, if necessary.
2. **Change** – Make any changes needed to the database definition or the application source code so that they function with the new version. If you are upgrading from Ingres 6.4, this step can be lengthy. If you are upgrading from a more recent version (for example, from Ingres II 2.0 to Advantage Ingres 2.6), few or no changes may be necessary.

All compatibility changes will be reflected back into the original version development installation. Thus, if the upgrade is delayed for some reason, no work will be lost.

3. **Test Applications** – Test your critical applications in the new-version development environment. Any problems or performance issues can be fixed before your production upgrade. The fix will nearly always be compatible with your original version as well, and therefore can be reflected back into your standard development environment.
4. **Practice the Upgrade** – When the application environment is ready, practice an upgrade using the test installation. Ideally, the test installation should be a duplicate of production. The trial upgrade should be repeated as often as necessary to achieve a trouble-free upgrade.

Tip: While practicing the upgrade, cease application development. You want the live upgrade to run exactly like the practice one, without involving new and untested factors.

5. **Perform the Upgrade** – Upgrade to the production system.

Application Issues

Before starting the upgrade, take an application and database inventory and make sure that every application can be rebuilt, starting from scratch. You must have complete and current source code for all applications because you will eventually recompile your applications under the new version. If the source code does not match what users are running, problems can result.

If you find an application that cannot be rebuilt, test the original executable under Ingres as soon as possible. If the application has no upward compatibility issues (for example, reserved words), it may be possible to run the old application against an Ingres installation and database. Otherwise, you will have to recreate the application or do without it.

Generally, try to synchronize the test and live Ingres upgrades with an appropriate time in the application life cycle. If application development is underway, you must plan how to coordinate new development with Ingres compatibility. Upgrades starting with newer versions (OpenIngres 1.2 or newer) may be able to move quickly enough to avoid any issue. Preparing an upgrade from Ingres 6.4 can take long enough to rule out a full stop in development.

One site, for example, addressed the timing issue by synchronizing Ingres compatibility with a code release. Then, development was converted to Ingres, while an Ingres 6.4 “bug fix” installation was maintained on a different machine.

The Test Plan for Applications

You must test your applications with the new version of Ingres before performing a production upgrade. The time and cost of testing every function in every application can be prohibitive. Fortunately, such testing is rarely necessary. A proper test plan can reduce testing time to as little as a week or two, even in the hardest case (upgrading from 6.4).

Keys to Success

Keys to a successful test plan are:

- Test only the most important parts of the application system.
- Fix problems found after the upgrade as quickly as possible.

The important question is not, “How important is this function?” but rather “How long can we live without this function?”

One successful testing approach divides application functions into three categories, as follows:

1. Functions that are business critical, and must be operational immediately after the upgrade. No delay is permitted.
Examples may include customer order entry, shipping, and production order release functions.
2. Functions that are important, but the business can survive their loss for a few hours after the upgrade.
Examples may include most inquiries and accounting functions, and high-visibility management reports, especially if management is made aware of the possibility of a one-time delay.
3. Functions that can be broken for a day or two without serious impact.
Examples may include reports, analysis functions, and end-of-period routines.

The first category of functions must be tested thoroughly. The second category should be tested as time and resources permit. The third category can usually be spot-checked.

Debugging

If you properly execute your test plan, all critical functions will work after the upgrade; less critical functions, however, may contain bugs. Be prepared to fix these bugs for a period after the upgrade. (Two weeks is usually long enough.) During this time, avoid scheduling new feature development. Have streamlined change control procedures ready, so that fixes can be installed quickly if a problem occurs.

Getting Started

This chapter describes how to move your existing development installation into a new Ingres development installation. In the new installation, you will test your databases and applications and make any necessary changes to ensure that they work correctly.

It describes how to:

- Create a development installation of the latest release of Ingres
- Prepare your applications
- Load your old version development databases and applications into the new development installation
- Prepare your system
- Test
- Practice the upgrade

The goal is to test thoroughly in the new development environment so that you can confidently perform the upgrade to the production system. The upgrade to the production system is described in subsequent chapters.

Note on Platform-specific Examples

While most of the examples used in this guide are specific to UNIX, the concepts described also apply to the Windows environment. For information on upgrading in the VMS environment, see the appendix “[Considerations for Alpha OpenVMS](#).”

Upgrading from Ingres 6.4

When migrating from Ingres 6.4, there are additional considerations not discussed in this chapter. Before performing the tasks in this chapter, see [Considerations for Ingres 6.4](#) in the appendix “Upgrading from Ingres 6.4.”

Creating a New Ingres Development Installation

This section describes how to install the latest release of Ingres into a development installation.

UNIX

The following procedure assumes that the development computer will support both the original and new version Ingres installations.

To install Ingres on the development machine, perform the following steps:

1. Create a new Ingres directory in a location with sufficient disk space. In this example, the directory is called `/ing26/ingres`.
2. Execute the following commands:

```
mkdir /ing26/ingres
```

```
chmod 755 /ing26/ingres
```

3. Set the environment to the original and new development installations. To do this, create two scripts; in this example, the scripts are named “setold” and “setnew.”

Here are example scripts for the C shell. You may need to adjust them for your specific installation.

For example, the PATH settings may be different, and LD_LIBRARY_PATH may be named LIBPATH or SHLIB_PATH, depending on the platform. In this example, the “old” installation is an Ingres 6.4 installation.

```
setold:
setenv II_SYSTEM /ing64
set path=(. /usr/local/bin /bin /usr/ucb /usr/sbin /usr/openwin/bin
$II_SYSTEM/ingres/bin $II_SYSTEM/Ingres/utility /usr/ccs/bin)
set inst=`ingprev1 II_INSTALLATION`
setenv LD_LIBRARY_PATH /usr/lib:/usr/openwin/lib
set prompt=`whoami`.`uname -n`[$inst]% "
echo "Switching to original Ingres 6.4 [$inst] installation"

setnew:
setenv II_SYSTEM /ing26
set path=(. /usr/local/bin /bin /usr/ucb /usr/sbin /usr/openwin/bin
$II_SYSTEM/ingres/bin $II_SYSTEM/ingres/utility /usr/ccs/bin)
set inst=`ingprev1 II_INSTALLATION`
setenv LD_LIBRARY_PATH /usr/lib:/usr/openwin/lib:$II_SYSTEM/ingres/lib
set prompt=`whoami`.`uname -n`[$inst]% "
echo "Switching to new 2.6 [$inst] installation"
```

4. If required, define aliases in the C shell or shell functions in the Bourne or Korn shell to invoke the setold and setnew scripts.

For example:


```
alias setold source ~ingres/setold
alias setnew source ~ingres/setnew
```

5. Use the “setnew” alias to switch to the new Ingres environment, and change directory to `$II_SYSTEM/ingres`.

Follow the installation instructions to install Ingres.

Note: Do not use the same data, checkpoint, journal, dump, or log directories as the original installation; the directories can, however, be on the same disks.

From Ingres 6.4 or 1.2

Take the time to investigate the new management facilities. Also, note that Ingres looks different in a “ps” listing. On a platform that supports OS threads or asynchronous I/O, there are no I/O slaves. The DMFRCP process, the Recovery Server, has been replaced by another IIDBMS process. 

Preparing Your Applications

While some applications created under your original version will run unchanged under the new Ingres, you will probably have to change some applications. None of the changes, however, is hard.

As part of the initial step of moving a copy of applications and databases to the new Ingres installation, you must check for:

- New reserved words
- Report-Writer syntax change, if upgrading from Ingres 6.4

New Reserved Keywords

A database cannot be built on an Ingres installation until reserved keyword conflicts are corrected.

Check for and fix reserved word conflicts. Also, check for reserved word conflicts in application code, specifically in dynamically created tables and views.

Ingres has a number of additional reserved keywords, mostly for support of the SQL additions. If words like *level*, *key*, or *comment* are used as column names, you must change them.

The SQL parser recognizes most reserved keywords from context, and usually resolves keyword conflicts without error, so you may not have to change reserved words used as names. If time permits, however, we recommend that you avoid SQL reserved words.

For a complete list of reserved words, see the appendix “[Keywords](#)” and the *SQL Reference Guide*.

Report-Writer Syntax Change

From Ingres 6.4

To support new Report-Writer syntax, a space is required after all dot-commands. For example, “.NL3” must be changed to “.NL 3”.

UNIX

To fix such occurrences automatically, you can use the following “sed” commands:

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

Compare the old and new files (foo.rw and newfoo.rw) to ensure that only the expected changes occurred. For example, you want to avoid an unwanted “fix” to a literal string.

An alternative to altering Report-Writer files is to **sreport** them into a database, then **copyrep** the reports back out.

Report-Writer Runtime Parameter Errors


UNIX

If string parameters that contain quotes are passed to Report-Writer, runtime errors may occur. These errors may be caused by a change to the UNIX command parameter control file utexe.def.

If such an error occurs, it is possible to change the command parameter back to the Ingres 6.4 utexe.def settings:

1. Edit \$II_SYSTEM/ingres/files/utexe.def
2. Search for the string "(%S)"
3. Change the string to: param '(%S)'

Save the file, retest, and see whether the error still occurs.

This problem only occurs with application systems developed under Ingres 6.4. However, you may need to check for the problem even if you are upgrading from a more recent version. Generally, the utexe.def file is replaced with every release of Ingres. Therefore, even if you have resolved this issue during a prior upgrade, you will have to check for it again each time you upgrade. 

Loading Databases and Applications into the New Installation

In this step, you move your development databases into the new Ingres development environment.

Creating Users

After your Ingres development installation is running, create any necessary Ingres users there. You may not need every user that exists in your current development environment. At a minimum, you must create any DBA (database owner) users.

Moving Databases

Note: Before performing this task, you should have already created procedures for switching between the original version and new version development environments, as described previously.

To move a database from the original development environment to your new Ingres environment, use a simplified unload/reload procedure, as follows:

1. **Setold** and **cd** to a directory with enough space to hold the data; allow for the Ingres System catalogs.
2. Create a directory for each database that is to be exported.
3. **Cd** to the directory for the database that is to be exported.
4. Execute **unloaddb** against the original-version database to be unloaded.
5. Execute **unload.ing** to export the front-end catalogs and data.
6. Edit the unload scripts as follows:

From Ingres 6.4

Edit the **cp_ingre.in** file and remove the lines:

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

Directory paths may be different.

From Ingres 1.2

Edit the **copy.in** file and remove the lines:

```
\include /ing12/ingres/files/iiud.scr  
\include /ing12/ingres/files/iiud65.scr
```

Directory paths may be different.

7. Fix the *system_maintained* column name if necessary. For more information, see [The system_maintained Column Name](#) in this chapter.

8. **Setnew** to the Ingres installation.
9. Create the database there, without any front-end catalogs, as follows:

```
createdb dbname -f nofeclients
```
10. If the Ingres database name is not the same as the original database name, then edit the reload.ing script.
11. Execute reload.ing for that database.

Tip: When using the reload.ing script, capture the output to a file in case any errors occur. On UNIX you can do this using “tee,” as follows:
reload.ing | & tee /temp/reload.log

Be sure to review the output of the reload for any reserved word conflicts. Correct any problems in the original-version environment and try again.

Moving Catalogs

At this point, the front-end catalogs in the Ingres database are in the original-version format. To put them into new-version format, run:

```
upgradefe <dbname> INGRES
```

The above assumes that you want the catalogs **and data** to be copied from the original development database to Ingres. If the data is not wanted, you can edit the scripts so that unloaddb does not copy certain tables.

Moving Distributed Option Databases

For Ingres Distributed Option (formerly Ingres Star) databases, **unloaddb** the CDB (the coordinator database, which usually starts with ii). This process unloads any locally stored tables that do not exist in other local databases.

Then **unloaddb** on the DDB (the distributed database, usually accessed by ddbname/star). This process unloads registrations and distributed view definitions.

The system_maintained Column Name

Databases created in releases prior to Ingres II 2.5 that contain the Metaschema module of system catalogs require an additional task when upgrading to Ingres II 2.5 using unload/reload.

These databases contain an extended system catalog *ii_atttype* with a column named *system_maintained*. As of Ingres II 2.5, *system_maintained* is a reserved word. Because of the keyword restriction, loading such a database into version 2.5 will fail. Release 2.6 and higher have a context sensitive keyword recognizer, and does not have the problem.

In Ingres II 2.5, the name of the *system_maintained* column is changed to *sys_maintained*. For the reload to work with 2.5, you must edit the original *copy.in* script to use the new column name. While you can also make this change using a utility such as *sed*, beware of inadvertently changing other uses of the *system_maintained* keyword.

Databases created with OpenIngres 1.x, Ingres version 6.4, and older do not usually contain the *ii_atttype* catalog. If you unload/reload a 6.4 database containing *ii_atttype*, you have to manually edit the file *cp_ingre.in* and fix *system_maintained* to *sys_maintained*.

Compiling Applications

After you have successfully imported your databases into the Ingres development environment, you must compile your applications in that environment. In most cases, you will want to make a copy of the application source code and libraries. Make sure that any compile scripts, linker command files, and the like point to the Ingres development installation, not the original development installation.

Testing

When you can successfully compile your applications with Ingres, you are ready to start testing. If you are upgrading from Ingres 6.4, there are additional application issues that you should check for, discussed in [Considerations for Ingres 6.4](#) in the appendix “Upgrading from Ingres 6.4.”

Preparing Your System

Some Ingres upgrade issues involve system or Ingres administration. Coordinate these changes with the system administrator.

System Monitoring Shellscripts

Production systems may have tools to provide the system administrator with early warning of Ingres problems. If these tools have been developed in-house, they must be reviewed for compatibility with your new Ingres release.

Some items to check for are:

- Are there still IO slaves on the UNIX platform? OS-thread architectures such as Windows and Sun Solaris do not use IO slaves.
- Does the tool parse iimonitor, logstat, or lockstat output? The detailed wording and positioning of logstat and lockstat output can change from release to release. Consider using IMA instead.
- If you are upgrading from Ingres 6.4, the log files II_RCP.LOG and II_ACP.LOG are renamed to iircp.log and iiacp.log.
- If your tool parses Ingres 6.4 parameters, you will have to change it. Ingres parameters are held in the files config.dat and protect.dat.
- If you are using a commercial monitoring tool, contact the vendor to see if an upgrade is needed to support Ingres.

If your Ingres monitoring tool uses the Ingres Monitoring Architecture (IMA), it is likely to continue to function with new Ingres versions. IMA is the recommended data source for any Ingres monitoring tool.

Checkpoint Template Changes

The Ingres checkpoint template file, cktpl.def, may change from release to release. If you have customized your checkpoint template file, you must review and verify your changes with the new Ingres version.

Ingres 6.4 or
OpenIngres 1.2

If you are upgrading from Ingres 6.4, or from OpenIngres 1.2, you must redo your template changes. The cktpl.def file format has been expanded since Ingres 6.4 and is therefore not compatible. The OpenIngres 1.2 template file format is similar to the current one, but additional entries are required. Your old checkpoint template can serve as a guide.

For more information on the new format of the checkpoint template file, see the chapter “Backup and Recovery” in the *Database Administrator Guide*.

Tip: If your checkpoint template was customized to do multiple location checkpoints in parallel, you may be able to remove this customization entirely. Ingres supports parallel checkpoint and rollforwarddb processing directly.

Ingres II

If you are upgrading from a more recent Ingres II release, compare your revised checkpoint template against the one installed with your new Ingres version. You may be able to use your customized template as is, but first check for new or changed entries in the new version.

The Ingres development installation can be used to develop and test the new cktmpl.def.

Other Checkpoint and Rollforward Changes

Typically, checkpoints and journals are not compatible from one version to the next. After an installation is upgraded, you must assume that all old checkpoints and journal files are no longer usable with the new version of Ingres.

Rollforwarddb no longer supports a **-b** option. (In Ingres 6.4, the **-b** option gave a starting time for applying journals.) Rollforwarddb no longer supports the **-noblobs** option as it makes the table physically inconsistent and unusable.

Backup and Restore

When upgrading, it is important to have a system backup. If something goes wrong, you will be able to restore from the backup.

Make sure that the system administrator knows how to take a complete system backup and how to restore that backup. Do a trial backup and verify that the backup is readable. This is especially important with tapes: failing tape drives can appear to write tapes without error, but the tapes may not be readable.

The system administrator should ensure that proper backup procedures are being followed. Backups taken as part of an upgrade should be removed from any backup media recycling, and kept in a secure location for a long time.

Shared Library Search Path

On many UNIX platforms, Ingres uses shared libraries. Since there is no default installation directory for Ingres, it is necessary to tell applications and tools where Ingres is installed so that the shared libraries can be found.

Use **one** of the following two ways:

- For all users who access any Ingres programs or applications, set the environment variables `LIBPATH`, `LD_LIBRARY_PATH`, or `SHLIB_PATH`, to include the Ingres library directory, `$II_SYSTEM/ingres/lib`.

Failure to have `LD_LIBRARY_PATH` set will result in an error message:

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

You can arrange for this setting ahead of time, while you are still running Ingres 6.4. The 6.4 binaries do not use `LD_LIBRARY_PATH`.

The exact name of the environment variable depends on your flavor of UNIX. Most UNIX platforms use `LD_LIBRARY_PATH`; HP-UX uses `SHLIB_PATH`; AIX versions 3 and 4 use `LIBPATH`. See the `ld(1)` or `ld.so(1)` man page in your operating system documentation.

- Link the Ingres library files to a standard UNIX library directory, such as `/usr/lib`.

For example:

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

and repeat for each `.so` file in the Ingres `lib` subdirectory.

This does not require application wrappers or user environment changes. The disadvantage of this approach is that you have to link (or copy) each Ingres library individually, and after a subsequent upgrade, check the validity of these links.

UNIX Kernel Parameters

Review the UNIX kernel parameter settings, particularly the maximum shared memory size.

If upgrading from Ingres 6.4, you may have to increase the size of a shared memory segment because Ingres builds a larger shared memory segment for locking and logging than did Ingres 6.4.

A 100 MB shared memory segment will accommodate most migrated installations. Each platform has its own way of modifying the shared memory limits; discuss this with the system administrator or read the platform-specific information in the `Readme` file.

If upgrading from a more recent version of Ingres, you probably do not have to change the kernel parameters. It is prudent to configure your new Ingres development installation similar to the production installation, to make sure that no kernel changes are needed.

Testing

As changes are made to the application for Ingres compatibility, bring the changes over to the development Ingres installation and test your applications according to your test plan.

When testing, use data that is as close to live data as possible. Performance critical functions should be tested against production data volumes.

Performance Testing

Include performance testing in your test plan. Changes to the query optimizer can cause queries to perform differently from your original Ingres version.

Typically, queries are faster, but can be slower in some cases. This is likely when a query has been tuned to work well with a peculiarity of the old-version query optimizer. If you notice performance problems, use the **set qep** command or the QEP display of Visual DBA. For more information on the use of query plans and optimizer statistics, see the *Database Administrator Guide*.

System Administrator Procedures

You should also test your system administration procedures. Crash test the Ingres installation when it is busy by pulling the power plug or issuing a system command to crash the servers. Make sure that recovery occurs correctly. Do at least one rollforwarddb of the most important databases and make sure it works in your environment.

Practicing the Upgrade

Run a trial upgrade as early as possible in the conversion cycle. Ideally, you should run trial upgrades more than once, so an isolated environment is desirable.

Take notes on what went wrong or what should be done differently. Continue practice upgrades until no more problems are encountered. Give the annotated upgrade procedure to someone who can verify the upgrade plan.

At least one of the practice upgrades should be on a full live data set so that you have an indication of how long the upgrade will take. This is particularly important when doing an unload/reload upgrade. In contrast, upgradedb type upgrades are largely insensitive to the amount of data in the database.

Before the Live Upgrade

As the date for the live upgrade approaches, freeze all changes, delete **all** application objects and images from the development Ingres installation, and re-image everything. Use this refreshed copy for, at least, a critical functions test.

Use this build for the live upgrade.

OpenROAD 4.0 Image File Formats

The image file format (.img) for OpenROAD images changed from OpenROAD 3.5 to OpenROAD 4.0. Image files built using the two releases are not compatible. Attempting to run an image from a different version will produce unpredictable results. OpenROAD 3.5 applications should be re-imaged for use under OpenROAD 4.0.

Upgrading Using Upgradedb

This chapter describes how to use the `upgradedb` utility to upgrade from a post-6.4 version of Ingres.

Upgrading using `upgradedb` transforms your database in-place from the original version to the new, without requiring an unload and reload. Unlike upgrading from 6.4, the internal database changes needed are relatively minor, so the upgrade process leaves the databases largely untouched.

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

Upgradedb Upgrade Procedure

Use the following procedure to perform an `upgradedb` upgrade from OpenIngres, Ingres II, or Advantage Ingres 2.6.

Procedure Notation

In this procedure, the notation **[Each DB]** means: “For each database, not including the `iidbdb`, become the DBA for that database and perform this step.”

Do not include the `iidbdb` or Ingres Distributed Option distributed databases unless instructed. If using the Distributed Option, remember to include the coordinator database in the list of databases.

Step 1: Disable User Access

Disable user access.

During the upgrade, the production system is not available for use. Make sure that users are not able to access the databases until the upgrade is complete.

Step 2: Disable Remote Command Server

The Remote Command Server component of Visual DBA must be disabled for the duration of the upgrade. The Remote Command Server uses the iidbdb database as a communications mechanism in versions of Ingres prior to 2.6, so it will interfere with upgrading.

To disable the Remote Command Server, run Configuration-By-Forms. Locate the row for the Remote Command Server, and note the startup count. Record this value for later, and then use the EditCount function to set the startup count to zero.

Note: If you are upgrading from early versions of OpenIngres 1.x, you may not see an entry for Remote Command Server. Skip this step.

Step 3: Shut Down Ingres and Back Up System

Perform a clean shutdown of Ingres, clearing all transactions from the transaction log. To achieve this, shut down Ingres, restart Ingres, and then shut it down again. Check the recovery process log (iircp.log) for the message "RCP Shutdown completed normally."

Take a system backup using a command appropriate to the platform. Make sure you include all Ingres directories: data, checkpoint, journal, dump areas, and the \$II_SYSTEM/ingres directory containing Ingres files and executables. Back up the application directories.

Watch for symbolic links and cross-mounts; make sure real data is saved and not a symbolic link.

If Ingres is typically started up at boot time, include the root file system in the backup. Alternatively, print a copy of any Ingres boot time startup and shutdown scripts.

For maximum safety, perform this step twice. At minimum, after the backup completes, check the backup media to ensure that the backup can be read. If your backup medium is tape, use new tapes, and clean the tape drive before the backup.

After completing the backup, restart Ingres.

Step 4: [Each DB Including iiddb] Clean the Database

Perform the following steps to ensure integrity of the system catalogs:

```
sysmod dbname
```

```
verifydb -mreport -sdbname dbname -odbms_catalog
```

The verifydb command may issue messages:

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 can ignore these messages. Also, ignore the “patch warning” message that warns of the loss of user tables in “runinteractive” mode. This mode will not be used.

If verifydb issues other warnings or errors, there may be damage to the system catalog. Before upgrading that database, review the messages with Computer Associates Technical Support.

Step 5: [Each DB] Record Database Information

Run infodb against each database, saving the output. The output will be needed later. You will need to know, for example, whether the database was journaled, where the database resides, and in what order the data locations were configured.

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

Step 6: [Each DB Including iiddb] Checkpoint and Turn Off Journaling

Checkpoint each database, using the ckpdb command with -j option to turn off journaling.

If upgradedb fails, you can use this checkpoint to recover and try again.

Save the configuration file stored in the dump area after each checkpoint. The configuration file is small.

To do this, issue these commands:

```
ckpdb -d -j dbname
```

```
cp $II_DUMP/ingres/dmp/default/dbname/aaaaaaaa.cnf {somewhere secure}
```

Step 7: Shut Down Ingres

Shut down Ingres with the `ingstop` command.

Step 8: Preserve Site Modifications

If you have customized any files that are distributed as part of Ingres, you must copy them because they will be lost during the upgrade. Any custom files you have **added** to the `$II_SYSTEM` directory tree will remain.

Commonly, customizations are made to the termcap and keyboard map files in `$II_SYSTEM/ingres/files`. Customizations are also made to local collation sequence files. Save the original collation definition files and the compiled files in `$II_SYSTEM/ingres/files/collation`.

Copy customized files to a safe place. Do **not** copy them to `/tmp` or anywhere in `$II_SYSTEM/ingres` directory.

If you cannot identify all your customized files, you can do the following to ensure that you preserve the necessary files:

1. Delete all `*.log` files from `$II_SYSTEM/ingres/files`
2. Copy to a safe place the entire contents of the following directories:
 - all `.opt` files
 - `$II_SYSTEM/ingres/bin`
 - `$II_SYSTEM/ingres/files`
 - `$II_SYSTEM/ingres/rep`
 - `$II_SYSTEM_ingres/files/rep`
 - `$II_SYSTEM/ingres/files/dayfile`
 - `$II_ingres/files/startup`
 - `$II_SYSEM/ingres/files/startsql`
 - `$II_SYSTEM/ingres/utility`

This procedure copies more files than necessary, but you can delete the copy after Ingres has been running live for a period.

Do not delete the copy immediately when the upgrade completes, because you may discover weeks later that you need the old version of a file (for example, a Vision template or keyboard map) from the original `$II_SYSTEM/ingres` directory.

UNIX

On UNIX, to copy these files, use commands similar to the following:

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

Visual DBA Configurations

When upgrading, Visual DBA configuration files (.vdbacfg) are not upwardly compatible and must be recreated.

Note that instead of using configuration files, you can use the `vdba` command with command line flags to start Visual DBA with, for example, certain windows open on given nodes. For details on the `vdba` command, see the *Command Reference Guide*.

Step 9: Delete Install Directory

Note: This step is only necessary on UNIX. It is optional but recommended.

UNIX

The Ingres installation procedure on UNIX starts by extracting the install subdirectory from the Ingres distribution.

You should delete the old contents of that directory first, as follows:

```
cd $II_SYSTEM/ingres
rm -rf install
```

Step 10: Install Ingres

To install Ingres, see the Ingres installation instructions for your platform.

During the installation process, the DBMS Server setup asks whether all databases are to be upgraded; answer **No**. The installation procedure automatically upgrades the `iidbdb`. If the upgrade of `iidbdb` fails, see the appendix “[Troubleshooting Upgradedb Problems](#).” It is better to complete the Ingres setup, and then use the `upgradedb` command to upgrade the user databases.

After the `iidbdb` is upgraded, the DBMS Server setup attempts to upgrade `imadb` and install Remote Command Server objects into `imadb`. Some versions of `upgradedb` neglect to create `imadb` first, and you will get “Database does not exist: `imadb`” errors. These will be corrected in the next step.


Upgrading to Older Versions That Require a Patch

Newer versions of Ingres distribute service packs. You can install service packs without having to install a base release of Ingres first.

UNIX

If you are upgrading to an older Ingres version that requires an overlay patch instead of a service pack, follow this procedure:

1. Run `ingbuild`. When asked whether you want to set up all the Ingres components, respond **No**. Exit `ingbuild`.
2. Install the Ingres patch.
3. Run `ingbuild` again. Select **Current**, then **SetupAll**.
4. Follow the prompts to complete the Ingres setup.


Setup now uses the fixed version. 

Step 11: Create imadb Database


Note: Perform this step only if you received “Database does not exist: imadb” messages during the DBMS setup phase of your Ingres install. This should only occur if you are upgrading from OpenIngres version 1.x.

As the installation owner, execute these commands:

UNIX

```
ingstart
cd $II_SYSTEM/ingres/vdba
createdb '-u$ingres' imadb -f nofeclients
sql '-u$ingres' imadb <makimau.sql
rmcmdgen
ingstop 
```

Windows

```
ingstart
cd %II_SYSTEM%\ingres\vdba
createdb -u$ingres imadb -f nofeclients
sql -u$ingres imadb <makiman.sql
rmcmdgen
ingstop 
```

As the `makimau` or `makiman` SQL scripts run, you see a series of messages such as “E_US0AC1 '*some-name*' does not exist or is not owned by you.” These are normal and can be ignored.

Step 12: Restore Site Modifications

Refer to the save directory that was created in Step 8 (see [Step 8: Preserve Site Modifications](#) in this chapter), and review any site-specific files that were overwritten by the upgrade.

Checkpoint Template If the checkpoint template file `cktmpl.def` has been modified, the modifications may need to be carried forward into Ingres. Your original `cktmpl.def` should not be used directly, as entries can be added or revised in new versions of Ingres. Compare your customized `cktmpl.def` with the newly installed file, and make necessary changes in the new `cktmpl.def`. For information about the checkpoint template, see the *Database Administrator Guide*.

Step 13: Start Ingres

Run `ingstart` to start Ingres.

Step 14: Run Upgradedb Utility

Run the `upgradedb` utility to upgrade databases.

You can upgrade databases one at a time or all at the same time. Log the `upgradedb` output to a file.

To upgrade one at a time:

```
upgradedb dbname
```

To upgrade all at the same time:

```
upgradedb -all
```

Example of logging `upgradedb` output to a file:

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

Troubleshooting If errors occur, see the appendix “[Troubleshooting Upgradedb Problems](#).” Correct the errors and rerun the `upgradedb` utility.

If the `upgradedb` command starts and then hangs with no error indication, the Remote Command Server may be interfering with the process. This is particularly likely if you are upgrading to Ingres II version 2.0 instead of Ingres. To fix, stop the Remote Command Server, as follows:

```
rmcmdstp
```

You can use Configuration-By-Forms or Visual Configurator to turn off the Remote Command Server until the upgrade is finished: select Remote Command Server and use `EditCount` to set the startup count to zero.

Step 15: Review Ingres Configuration

The upgrade preserves your original Ingres installation parameters. You should review the configuration because some parameters may change from version to version. For information on parameters that changed, check the Readme for your new version of Ingres.

Review your parameter settings by running Configuration-By-Forms or Visual Configurator. Especially pay attention to major items such as startup counts and DBMS cache settings.

Note: If you disabled the Remote Command Server in Step 2 (see [Step 2: Disable Remote Command Server](#) in this chapter), use EditCount to restore its startup count to the original value.

Step 16: [Each DB] Reapply Optimizer Statistics (Optional)

Note: This step is required only if upgrading from OpenIngres 1.x. The step is optional. Ingres computes additional metrics that OpenIngres 1.x did not have.

To take advantage of the new metrics, regenerate the optimizer statistics using the procedures of your application system.

Step 17: [Each DB including iidbdb] Checkpoint the Database

Checkpoints and journals from your original Ingres version will not work with Ingres, so do not omit or delay this step.

Checkpoint each database. If the database was journaled previously, use the +j flag to turn on journaling.

To know which databases were journaled, see the infodb output from Step 5 (see [Step 5: \[Each DB\] Record Database Information](#) in this chapter).

The iidbdb should always be journaled, regardless of whether it was journaled in the original installation.

Step 18: Install Upgraded Applications

Install the Ingres versions of the applications. Then restore user logins and resume normal operation.

Upgrading Using Unload/Reload

This chapter describes how to use the unload/reload procedure to upgrade from a post-6.4 version of Ingres.

The unload/reload upgrade avoids the `upgradedb` program (except for `iidbdb`), in favor of unloading the original Ingres databases to flat files, recreating the databases under Ingres, and then reloading the databases. This approach has the advantage of starting with clean databases, but requires more time and disk space than does the `upgradedb` method.

Note: Databases using the system-maintained logical key feature are best upgraded using `upgradedb`. Tables that contain `SYSTEM_MAINTAINED` `table_key` or `object_key` columns cannot be safely unloaded and reloaded without additional work. The reload step generates all new logical key values. If other tables reference the logical key columns, the new values must be manually propagated to those tables.

Two Variations

The unload/reload procedure has two variations:

- The **in-place upgrade**, which replaces the original installation with the new Ingres installation. The master database (`iidbdb`) is upgraded with `upgradedb`, even though other databases are unloaded and reloaded. Because the `iidbdb` remains, all your locations, users, groups, and roles still exist in the new installation.
- The **clean install upgrade**, which leaves the original installation alone. Ingres is installed into a completely new installation. (The new installation may even be on a different machine.) When performing a clean install upgrade, you must take extra steps to recreate locations and move users, groups, and roles from the original installation to the new one.

Unload/Reload Upgrade Procedure

Use the following procedure to perform an unload/reload upgrade from OpenIngres, Ingres II, or Advantage Ingres 2.6.

Procedure Notation In this procedure, the notation **[Each DB]** means: “For each database, not including the iidbdb, become the DBA for that database, **cd** to the unload directory for the database created in Step 1, and perform this step.”

If using the Distributed Option, include the coordinator database in the list of databases.

Steps that apply to a particular upgrade type only (that is, in-place upgrade or clean install upgrade) are marked accordingly.

Step 1: [Each DB Including iidbdb] Create Unload Directories

Create a directory for each database. The directory is used to hold scripts and data from the unloaded database and requires a large amount of disk space. As an estimate, the unloaded data is about the same size as the Ingres database; however, compressed data can expand to take up much more space than the Ingres database.

UNIX

On UNIX, use these commands:

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

Windows

On Windows, use this command:

```
mkdir d:\someplace\dbname
```

Step 2: [Each DB] Run Unloaddb

Run unloaddb against each database. The unloaddb command does not unload the database; it simply creates copy in and copy out scripts.

Note: For Distributed Option databases, unload the CDB in the same way as for a local database. For a DDB, use unloaddb/star.

For a regular DB or CDB:

```
unloaddb dbname
```

For a Distributed Option DDB:

```
unloaddb ddbname/star
```

If doing a clean-install upgrade to a different machine that has a newer architecture, binary data may not be compatible between the two machines. If this is the case, use the `unloaddb -c` option, which causes an ASCII instead of binary unload.

Step 3: [Each DB] Check for Obsolete Users

Old databases may have objects created by users who no longer exist.

Examine the `copy.out` and `copy.in` scripts created by `unloaddb` in Step 2. Each script contains **set session authorization** SQL statements for each user who owns a database object. Search for the **set session authorization** statements, and make sure that all users listed are valid.

If obsolete users are found, delete all the lines from that **set session authorization** statement up to the next one. Also, go into the database and clean out these unwanted objects.

Step 4: [Each DB Including iidbdb] Checkpoint the Database (Optional)

Note: This step is optional. You can omit this step if you can rely on the system backup to be taken in Step 7 (see [Step 7: Shut Down Ingres and Back Up System](#) in this chapter).

Checkpoint each database and then copy the checkpoint files to a permanent medium such as tape. If using tape, use fresh tape and verify that the tape can be read.

Remember to checkpoint the `iidbdb`.

Step 5: Disable User Access

Disable user access.

Step 6: Disable Remote Command Server

The Remote Command Server component of Visual DBA must be disabled for the duration of the upgrade. The Remote Command Server uses the iidbdb database as a communications mechanism in versions of Ingres prior to 2.6, so it will interfere with upgrading.

To disable the Remote Command Server, run Configuration-By-Forms. Locate the row for the Remote Command Server, and note the startup count. Record this value for later, and then use the EditCount function to set the startup count to zero.

Note: If you are upgrading from early versions of OpenIngres 1.x, you may not see an entry for Remote Command Server. Skip this step.

Step 7: Shut Down Ingres and Back Up System

Note: This step is required only for an in-place upgrade. However, this is a convenient time to get a complete backup of your production system before the upgrade. Therefore, you may want to perform this step even if doing a clean install upgrade.

Perform a clean shutdown of Ingres, clearing all transactions from the transaction log. To achieve this, shut down Ingres, restart Ingres, and then shut it down again. Check the recovery process log (II_RCP.LOG) for the message "RCP Shutdown completed normally."

Take a system backup using a command appropriate to the platform. Make sure you include all Ingres directories: data, checkpoint, journal, dump areas, and the \$II_SYSTEM/ingres directory containing Ingres files and executables. Back up the application directories.

Watch for symbolic links and cross-mounts; make sure real data is saved and not a symbolic link.

If Ingres is typically started up at boot time, include the root file system in the backup. Alternatively, print a copy of any Ingres boot time startup and shutdown scripts.

For maximum safety, perform this step twice. At minimum, after the backup completes, check the backup media to ensure that the backup can be read. If your backup medium is tape, use new tapes, and clean the tape drive before the backup.

After completing the backup, restart Ingres.

Step 8: [Each DB] Unload the Database

For each database, run the unload.ing script created by unloaddb. This unloads the database into your unload directory.

Step 9: [Each DB] Print Optimizer Statistics (Optional)

Note: This step applies only to a clean-install upgrade.

If your upgrade plan allows enough downtime to run a full optimizedb against your databases, you can omit this step. If your plan does not allow enough downtime, perform this step as a shortcut.

Note: Be aware that using this shortcut may result in some of the new Ingres metrics not being available; query performance may suffer until a full optimizedb can be completed.

If you are upgrading from OpenIngres 1.x, you should regenerate new statistics instead of saving the old ones, if possible.

To dump the existing optimizer statistics, run statdump with the -o flag to a file for each database, as follows:

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

Step 10: [Each DB] Record Database Information

Run infodb against each database, saving the output. The output will be needed later. You will need to know, for example, whether the database was journaled, the data locations where the database resides, and in what order the locations were configured.

Run infodb as follows:

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

Also, record whether the database is public or private. To find out, use the catalogdb command. Select Databases, and then enter the database name. The screen that appears has an Access field that indicates whether the database is public or private.

Step 11: Record Database Privileges

As the installation owner, change directories to the unload directory for iiddb created in Step 1. Save user database privileges, as follows:

```
sql iiddb
\script dbprivs.out
select *
from iiddbprivileges
where database_name <> ''
order by database_name,grantee_name
\go
\script
\quit
```

This procedure creates the file dbprivs.out for future reference.

Step 12: Save Users, Groups, and Roles

Note: This step is required only for a clean-install upgrade.

As the installation owner, change directory to the iiddb unload directory, as you did in the previous step. Run the following SQL to save users, groups, and roles:

```
sql iiddb
copy iiusergroup (
    groupid=c0comma,groupmem=c0nl
) into 'groups.out'
\go

copy iirole(
    roleid=c0nl
) into 'roles.out'
\go

create table role_tmp as
select role_name,grantee_name
from iirolegrant
where admin_option <> 'Y'
\go
copy role_tmp(
    role_name = c0comma,
    grantee_name = c0nl
) into 'rolegrants.out';
drop role_tmp;
\go
\quit
```

Next, run accessdb. Select Users, then SqlScript. This writes a file called users.sql that will recreate all users, as they are currently defined.

Step 13: [Each DB] Destroy the Database

Note: This step is required only for an in-place upgrade.

Destroy each database using the `destroydb` command.

Step 14: Clean iidbdb Database

Note: This step is required only for an in-place upgrade.

Become the installation owner and run the following steps against the master database `iidbdb`. It is assumed that there are no objects created by users in the `iidbdb`.

```
sysmod iidbdb
verifydb -mrun -sdbname iidbdb -opurge
verifydb -mrun -sdbname iidbdb -odbms
ckpdb -j iidbdb
```

The `verifydb -odbms` command may issue the following messages:

`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 can ignore these messages. Also, ignore the “patch warning” message that warns of the loss of user tables in “runinteractive” mode. This mode will not be used.

Step 15: Shut Down Ingres

Shut down Ingres with the `ingstop` command.

Step 16: Disable Ingres Startup

Note: This step is recommended even if you are doing a clean installation upgrade. By leaving the old 6.4 installation shut down, you eliminate the chance that someone will connect to it by mistake later.

If Ingres starts automatically when the machine boots up, turn auto-starting off until the upgrade is complete.

On most UNIX platforms, a file in a system startup directory performs Ingres startup and shutdown; place an “exit 0” at the top of this file. The system administrator may need to perform this step if it requires root privilege. (The system startup directory depends on your platform — /etc/init.d, or /sbin/init.d, or a similar name).

On Windows, if Ingres is run as a system service, set the service to start manually instead of automatically.

Make sure that the operating system is correctly configured for your new version of Ingres (see [Preparing Your System](#) in the chapter “Getting Started”).

Reboot, if necessary, to put the operating system parameter changes into effect.

Step 17: Preserve Site Modifications

If you have customized any files that are distributed as part of Ingres, you must copy them because they will be lost during the upgrade. Any custom files you have **added** to the \$II_SYSTEM directory tree will remain.

Commonly, customizations are made to the termcap and keyboard map files in \$II_SYSTEM/ingres/files. Customizations are also made to local collation sequence files. Save the original collation definition files and the compiled files in \$II_SYSTEM/ingres/files/collation.

Copy customized files to a safe place. Do **not** copy them to /tmp or anywhere in \$II_SYSTEM/ingres directory.

If you cannot identify all your customized files, you can do the following to ensure that you preserve the necessary files:

1. Delete all *.log files from \$II_SYSTEM/ingres/files
2. Copy to a safe place the entire contents of the following directories:
 - all .opt files
 - \$II_SYSTEM/ingres/bin
 - \$II_SYSTEM/ingres/files
 - \$II_SYSTEM/ingres/rep
 - \$II_SYSTEM_ingres/files/rep
 - \$II_SYSTEM/ingres/files/dayfile
 - \$II_ingres/files/startup
 - \$II_SYSEM/ingres/files/startsql
 - \$II_SYSTEM/ingres/utility

This procedure copies more files than necessary, but you can delete the copy after Ingres has been running live for a period.

Do not delete the copy immediately when the upgrade completes, because you may discover weeks later that you need the old version of a file (for example, a Vision template or keyboard map) from the original \$II_SYSTEM/ingres directory.

UNIX

On UNIX, to copy these files, use commands similar to the following:

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

Visual DBA Configurations

When upgrading, Visual DBA configuration files (.vdbacfg) are not upwardly compatible and must be recreated.

Note that instead of using configuration files, you can use the vdba command with command line flags to start Visual DBA, for example, with certain windows open on given nodes. For details on the vdba command, see the *Command Reference Guide*.

Step 18: Delete Install Directory

Note: This step is required only for an in-place upgrade on UNIX. It is optional but recommended.

UNIX

The Ingres installation procedure on UNIX starts by extracting the install subdirectory from the Ingres distribution.

You must delete the old contents of that directory first, as follows:

```
cd $II_SYSTEM/ingres
rm -rf install
```

Step 19: Install Ingres

To install Ingres, see the Ingres installation instructions for your platform.

In-place upgrades only: During the installation process, the DBMS Server setup asks whether all databases are to be upgraded; answer **No**. The install procedure automatically upgrades the iidbdb. If the upgrade of iidbdb fails, see the appendix “[Troubleshooting Upgradedb Problems](#).”

After the iidbdb is upgraded, the DBMS Server setup attempts to upgrade imadb and install Remote Command Server objects into imadb. Some versions of upgradedb neglect to create imadb first, and you will get “Database does not exist: imadb” errors. These will be corrected in the next step.


Upgrading to Older Versions That Require a Patch

Newer versions of Ingres distribute service packs. You can install service packs without having to install a base release of Ingres first.

UNIX

If you are upgrading to an older Ingres version that requires an overlay patch instead of a service pack, follow this procedure:

1. Run ingbuild. When asked whether you want to set up all the Ingres components, respond **No**. Exit ingbuild.
2. Install the Ingres patch.
3. Run ingbuild again. Select Current, then SetupAll.
4. Follow the prompts to complete the Ingres setup.

Setup now uses the fixed version. 


Step 20: Create imadb Database

Note: Perform this step only if you received “Database does not exist: imadb” messages during the DBMS Setup phase of your Ingres install.


Create the imadb database.

As the installation owner, execute these commands:

UNIX

```
ingstart
cd $II_SYSTEM/ingres/vdba
createdb '-u$ingres' imadb -f nofeclients
sql '-u$ingres' imadb <makimau.sql
rmcmdgen
ingstop 
```

Windows

```
ingstart
cd %II_SYSTEM%\ingres\vdba
createdb -u$ingres imadb -f nofeclients
sql -u$ingres imadb <makiman.sql
rmcmdgen
ingstop 
```

As the makimau or makiman SQL scripts run, you see a series of messages such as “E_US0AC1 'some-name' does not exist or is not owned by you.” These are normal and can be ignored.

Step 21: Restore Site Modifications

Refer to the save directory that was created in Step 17 (see [Step 17: Preserve Site Modifications](#) in this chapter), and restore any site-specific files.

Checkpoint Template If the checkpoint template file cktmp1.def has been modified, the modifications may need to be carried forward into Ingres. Your original cktmp1.def should not be used directly, as entries can be added or revised in new versions of Ingres. Compare your customized cktmp1.def with the newly installed file, and make necessary changes in the new cktmp1.def. For information about the checkpoint template, see the *Database Administrator Guide*.

Step 22: Review Ingres Configuration

If you are doing a clean install, you need to change the default Ingres configuration to match your site requirements.

If you are doing an in-place upgrade, the upgrade process preserves your original Ingres installation parameters. You should review the configuration because some parameters may change from version to version. For information on parameters that changed, check the Readme for your new version of Ingres.

Review your parameter settings by running Configuration-By-Forms or Visual Configurator. Especially pay attention to major items such as startup counts and DBMS cache settings. If you are doing a clean install, you can use your original Ingres installation configuration as a guide.

Note: If you disabled the Remote Command Server in Step 6 (see [Step 6: Disable Remote Command Server](#) in this chapter), use EditCount to restore its startup count to the original value.

Step 23: Set Up Ingres Net

Run netutil to create the vnode definitions for the remote installations. If installation passwords are needed, you must run mkvalidpw. See the *System Administrator Guide* or the Readme for your platform.

If there are NFS client-only installations that have not been set up, run ingmnfs to set them up.

Step 24: Start Ingres

Run `ingstart` to start Ingres.

Step 25: Recreate Users, Groups, and Roles

Note: This step is required only for a clean-installation upgrade.

As the installation owner, change directory to your `iidbdb` `unloaddb` directory where you stored the files from Step 12 (see [Step 12: Save Users, Groups, and Roles](#) in this chapter).

Run this SQL:

```
sql '-u$ingres' iidbdb
copy iusergroup(groupid=c0comma,groupmem=c0nl)
from 'groups.out'
\go
commit
\go
\read users.sql
commit
\go
\quit
```

Windows

For Windows, omit the quotes from the `sql` command line. ❗

The file `users.sql` may try to recreate some users that already exist in the installation, such as the installation owner and root user. This will cause “E_US18B6 The user '*name*' already exists” errors. You can ignore these errors.

If your original installation had roles defined, recreate them with the `ADD ROLE` SQL statement. Use the file `roles.out` as a guide. Roles cannot be reliably bulk-loaded from the original installation, so you must recreate them by hand. After you recreate each role, issue the following SQL statement:

```
grant rolename to user; commit
```

The most common *user* here is **public**. You can use the file `rolegrants.out` to determine what role grants are needed.

Step 26: Recreate Locations

Note: This step is required only for a clean-install upgrade.

Refer to **each** `infodb` output saved in Step 10 (see [Step 10: \[Each DB\] Record Database Information](#) in this chapter). Create any location that is not a default installation location (`ii_database`, `ii_checkpoint`, `ii_journal`, or `ii_dump`).

For more information about creating locations, see the *Database Administrator Guide*.

Step 27: [Each DB] Recreate the Database

Before creating each database, refer to the infodb output saved in Step 10 (see [Step 10: \[Each DB\] Record Database Information](#) in this chapter). Look at the location names for ROOT, JOURNAL, CHECKPOINT, and DUMP. If these are not ii_database, ii_journal, ii_checkpoint, or ii_dump, you must specify the location to createdb with the -d, -j, -c, or -b flags, respectively.

Also, refer to the database access information recorded in that step. If the database access was “private,” you must use the -p flag for createdb.

If all the database locations are the default, and the database is public, you can omit the flags on the createdb command line.

Recreate each user database, omitting the front-end catalogs. (The front-end catalogs will be created as part of the reload.) Use the following command:

```
createdb dbname flags -f nofeclients
```

Note: For a Distributed Option database, run createdb/star for the DDB. Do not run createdb for the CDB.

Step 28: [Each DB] Extend the Database

Refer to the infodb output saved in Step 10 (see [Step 10: \[Each DB\] Record Database Information](#) in this chapter). If the database was extended to data locations other than the default location, run accessdb as the installation owner and extend the newly-created databases to the same locations. The locations will already exist; it is only necessary to extend the databases to use them.

If you prefer a non-interactive command line utility, you can use the extenddb utility instead of accessdb.

Step 29: Recreate Database Privileges

As the installation owner, change to the iidbdb unloaddb directory, and refer to the file dbprivs.out created in Step 11 (see [Step 11: Record Database Privileges](#) in this chapter).

Each row describes one or more database privileges given to the user *grantee-name*. A Y or N in a privilege column indicates the specific privilege. (A U in a column means “Unchanged.”)

Start an iidbdb Terminal Monitor session:

```
sql iidbdb
```

For each row, issue the statement:

```
grant privilege on database database-name to grantee-name;commit
```

If the privilege column is N, grant *noprivilege* instead of *privilege*.

When finished, use **\quit** to exit the iidbdb session.

The structure of the iidbpriv catalog did not change between OpenIngres 1.x and Advantage Ingres 2.6, so it is possible to copy the original contents of the catalog directly. However, we do not recommend this because the catalog may change in future releases.

If you have defined many privileges, or recreated many users, groups, or roles, you should run sysmod on the iidbdb, which will accelerate query processing. Issue the sysmod command, as follows:

```
sysmod iidbdb
```

Step 30: [Each DB] Fix FE Reload Script

Edit the file copy.in and locate the lines:

```
\include/ing12/ingres/files/iiud.scr  
\include/ing12/ingres/files/iiud65.scr
```

Note: The directory path may differ.

Delete these lines. Because the new database was not created with front-end catalogs, it is not necessary to drop them.

Also, check for the *ii_atttype* catalog definition:

```
create table ii_atttype (  
.  
.  
...about 23 lines...  
.  
.  
        system_maintained char(1) not null
```

Change the name **system_maintained** to **sys_maintained**.

Not all databases contain the *ii_atttype* catalog, so it is okay if you do not find the definition.

Save the modified copy.in file.


Step 31: [Each DB] Reload the Database

Run reload.ing for each database.

UNIX

Redirect the reload to a log file so that it can be checked for errors. Using the C shell:

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

Note: If using the Distributed Option, reload the CDB and all “real” local databases before reloading the DDBs. 

After the reload is complete, verify that the table ii_id has only one row. Type **isql <database>**, and **select * from ii_id**. If more than one row is returned, delete the row with the lowest object_id.

Step 32: [Each DB] Upgrade Front-End Catalogs

Run upgradefe on each database, which brings the front-end catalogs up to Ingres level. Issue the following command:

```
upgradefe dbname INGRES
```

The word INGRES should appear in uppercase.

Step 33: [Each DB] Reapply Optimizer Statistics

Regenerate optimization statistics. You can do this either by regenerating statistics from scratch or by using the original statistics printed from the original installation earlier in this upgrade procedure (see [Step 9: \[Each DB\] Print Optimizer Statistics \(Optional\)](#)).

If there is sufficient time, we recommend that you regenerate the optimizer statistics using the procedures of your application system.

If time is short, and if you printed the original statistics in Step 8, you can read them back in with the -i option to optimizedb:

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

Step 34: [Each DB including iidbdb] Checkpoint the Database

Checkpoints and journals from your original Ingres version will not work with Ingres, so do not omit or delay this step.

Checkpoint each database. If the database was journaled previously, use the +j flag to turn on journaling.

To know which databases were journaled, see the infodb output from Step 10 (see [Step 10: \[Each DB\] Record Database Information](#) in this chapter).

The iidbdb should always be journaled, regardless of whether it was journaled in the original installation.

Step 35: Install Upgraded Applications

Install the latest Ingres versions of the applications. Then, restore user logins and resume normal operation.

Considerations for Alpha OpenVMS

This appendix describes the steps required for upgrading Ingres on the Alpha OpenVMS platform from Ingres II 2.0 to Advantage Ingres 2.6/0401 or Ingres r3 (axm.vms/00).

Use this chapter in conjunction with the appropriate edition of the Ingres *Getting Started* guide and the Readme file.

OpenVMS Requirements

For the minimum process requirements for an Ingres system administrator, see the appendix “System Requirements for OpenVMS” in the *Getting Started* guide.

Installing Ingres

The installation process has not changed significantly from Ingres II 2.0. For full instructions on installing Ingres on OpenVMS, see the *Getting Started* guide.

Ingres uses the VMSINSTAL procedure to install and configure its software. Using VMS, it is possible to create the new Ingres System Administrator account, extract all the software required, and configure Ingres. However, depending on how the installation progresses, some issues may develop.

Mounting the CD

You can install Ingres either directly from the CD-ROM or from a working area on the target system. If the files are transferred to the target node through FTP, they must be moved across in binary mode. However, if the machine has a CD-ROM drive, you can use the following command to mount the CD:

```
$ MOUNT /OVERRIDE=IDENTIFICATION /MEDIA_FORMAT=CDROM -  
/UNDEFINED_FAT=(FIXED:CR:32256) <CD Device>
```

To access the release notes use the following:

```
$ MOUNT /OVERRIDE=IDENTIFICATION /MEDIA_FORMAT=CDROM <CD Device>
```

Running VMSINSTAL

Ingres can be installed from any *privileged* account, defined as holding the privileges needed to run Ingres. (For the required Ingres privileges, see the *Getting Started* guide.)

To run the installer issue the command:

```
@sys$update:vmsinstal * distribution_medium
```

By default, the SYS\$ROOT area is used by VMSINSTAL to unpack the savesets in preparation for installing Ingres. If there is insufficient space available then VMSINSTAL will fail. To specify an alternate working directory, you can use the *awd* parameter, as follows:

```
@sys$update:vmsinstal * <distribution_medium> options awd=device:[dir]
```

To log the installation process specify the option *L* when calling VMSINSTAL:

```
@sys$update:vmsinstal * <distribution_medium> options L
```

Known Installation Issues

Note: For more information about these issues, contact technical support or check the technical documents available at the Computer Associates Technical Support web site, accessible from <http://ca.com>.

Creating the Ingres System Administrator account from within VMSINSTAL does not assign the correct process quotas to the account. (For the correct quotas, see the *Getting Started* guide.) The workaround is to create the account before the VMSINSTAL process is started with the correct privileges and process quotas.

II_WORK is not picked up, if pre-defined in the local symbol table, when an Express installation is performed. The user must enter the correct information when prompted.

If Ingres is installed from a non-Ingres System Administrator account, imadb is created as the process owner for VMSINSTAL rather than the installation owner configured earlier on. When Ingres is started, the RMCMD process will hang as it is running as a user that is unable to connect to the RMCMD catalogs in imadb. The workaround is to install Ingres as the intended Ingres System Administrator.

Schema Checking

Ingres reserves a number of new keywords, mostly for support of SQL additions. If names such as *substring*, *first*, or *cache* are used as column names, you must change the database schema. For a list of Ingres reserved words, see the appendix “[Keywords](#)” and the *SQL Reference Guide*.

If you are concerned that some column names in your database may conflict with reserved words, you can take a copy of the schema from the current installation and load it into an Ingres database. You should extend these checks to the applications to verify that tables and views created at runtime are not affected by the new keywords. Conflicts found in the schema and applications must be removed before moving to Ingres.

Rebuilding Applications

In addition to migrating data, you must rebuild all applications that connect locally to an existing Ingres installation.

The following compilers have been tested and are known to work with Ingres for OpenVMS. For the latest information, see the *Readme*:

HP Ada	HP BASIC
HP C	HP COBOL
HP C++	HP Pascal
HP Fortran	

Building Member_Aligned Against Ingres 2.6 or r3

Note: This section applies only to migrating from releases prior to Ingres II 2.0/0011 (axm.vms/00).

With the move to a member_aligned version of Ingres, some applications must be rebuilt. Applications that connect directly to an installation located on the same node, or through Ingres Net on the client node, must be rebuilt.

Building Vision and Application-By-Forms applications member_aligned is the default behavior, with no further changes being required from the developer.

C	/MEMBER_ALIGNED
Pascal	/ALIGN=ALPHA_AXP

COBOL	/ALIGNMENT=Padding
Fortran	/ALIGNMENT=ALL

If an application cannot be built using Alpha member alignment, it is possible to rebuild it with the Ingres components naturally aligned. The steps needed for C and COBOL applications are described in the following sections.

These changes require modification to the Ingres supplied files only and not the application code. Even by performing the steps listed here, you still must recompile **all** parts of the application that interface with Ingres or that use any structures declared in the Ingres header files.

By default, user applications are built using the same compiler options used to build the Ingres libraries and applications. If these options are not used, proceed carefully.

The introduction of the member_aligned version of Ingres came about when alignment-related memory issues were encountered in Ingres II 2.0/9808 (xvp.vms/00). If any applications are built using un-aligned structures with the communicating interface to Ingres, data corruption is likely to occur.

For C Applications

To build C applications byte-aligned with Ingres, a number of files require modification. Any modifications made may need to be re-applied following the installation of an Ingres patch.

The first files to modify are the C header files supplied in the following directories:

```
II_SYSTEM: [INGRES.DEMO.API.ASC]
II_SYSTEM: [INGRES.DEMO.UDADTS]
II_SYSTEM: [INGRES.FILES]
```

At this time, the only header files that contain Ingres structure definitions need modification, these are:

```
II_SYSTEM: [INGRES.DEMO.API.ASC]ASC.H
II_SYSTEM: [INGRES.DEMO.UDADTS]UDT.H
II_SYSTEM: [INGRES.FILES]ABFURTS.H,
II_SYSTEM: [INGRES.FILES]EQSQLCA.H,
II_SYSTEM: [INGRES.FILES]EQSQLDA.H,
II_SYSTEM: [INGRES.FILES]FRAME2.H,
II_SYSTEM: [INGRES.FILES]FRAME60.H,
II_SYSTEM: [INGRES.FILES]FRAME61.H,
II_SYSTEM: [INGRES.FILES]IIADD.H,
II_SYSTEM: [INGRES.FILES]IIAPI.H,
II_SYSTEM: [INGRES.FILES]OSLHDR.H,
II_SYSTEM: [INGRES.FILES]RAAT.H,
II_SYSTEM: [INGRES.FILES]SPATIAL.H
```

On the first line in each of the above files add:

```
#pragma member_alignment save  
#pragma member_alignment
```

On the last line in each of the above files add:

```
#pragma member_alignment restore
```

Note: The "#" of the #pragma instruction *must* be the first character on the line.

The purpose of these pragmas is to direct the compiler to naturally align the elements of the defined structures, then to restore the alignment strategy used before the header file was included.

One further change is required to allow Application-By-Forms and Vision applications to successfully build with unaligned code. In

II_SYSTEM:[INGRES.FILES]DCC.COM, replace the line

```
$ cc/standard=vaxc/float=ieee_float/nooptimize/nolist
```

with:

```
$ cc/NOMEMBER_ALIGNMENT/GRANULARITY=BYTE -  
/standard=vaxc/float=ieee_float/nooptimize/nolist
```

For COBOL Applications

To achieve the same result for embedded COBOL applications, the following statements must be added to these files:

```
II_SYSTEM:[INGRES.FILES]EQSQLCA.COB, II_SYSTEM:[INGRES.FILES]ESQLDA.COB
```

On the first line of the above files, add:

```
*DC SET ALIGNMENT
```

On the last line of the above files, add:

```
*DC END-SET ALIGNMENT
```

The II_SYSTEM:[INGRES.FILES]UTCOM.DEF file requires the removal of the qualifier "/alignment=padding" from the COBOL compile statements.

Upgrading from Ingres 6.4

Considerations for Ingres 6.4

This section describes additional considerations when loading Ingres 6.4 databases and applications into the new development installation.

Preparing Your Applications

Check for the following additional application issues after successfully creating databases and applications in the Ingres development installation.

UPDATE . . . FROM Semantics Change

In Ingres 6.4/05 and earlier, the “ambiguous replace” test allowed an update using the UPDATE...FROM statement if each target row was being updated with an unambiguous value. Ingres 6.4/06 and higher releases test for multiple FROM rows and generate an ambiguous replace error message even if all the FROM rows generate the same replacement value.

For example, Ingres 6.4/05 and earlier allowed the following update:

```
UPDATE    table_1
FROM      table_2
SET       column_3 = 3;
```

even though there is no WHERE qualification joining the tables, since the replacement value was non-ambiguous. In later releases, an “ambiguous replace” error message displays.

The recommended approach for this semantics change is to review all applications for ambiguous updates and change them to use EXISTS or IN, instead of a join. If this is not feasible, the original UPDATE . . . FROM handling can be restored by setting the DBMS parameter “ambig_replace_64compat” to ON in Configuration-By-Forms.

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.

Typically, this does not matter, as Ingres does appropriate type conversions. However, it is important when doing a CREATE TABLE . . . AS SELECT with a constant in the SELECT result list.

For example:

```
CREATE TABLE table_1
  AS SELECT column_1, column_2, column_3=1.0
  FROM table_2;
```

In Ingres 6.4, the column_3 is created as FLOAT8; in Ingres it is created as a DECIMAL(2,1) column. This may result in overflow in an application.

The recommended approach is to examine uses of fixed-point constant usage in applications and change them to floating point constants, or add an explicit FLOAT8 type conversion.

A less thorough but easier alternative is to set the environment variable II_NUMERIC_LITERAL to FLOAT, as follows:

```
setenv II_NUMERIC_LITERAL FLOAT
```

Ingres then interprets fixed-point constants as floats rather than decimals. If you decide to use II_NUMERIC_LITERAL, it will be necessary for **every** user of the applications to set II_NUMERIC_LITERAL in their environment.

Greater Sensitivity to BYREF Errors

Ingres 6.4 4GL programs are insensitive to length and type errors when returning BYREF values to a calling program. Ingres is more sensitive to return values that are too long or the wrong type. In some cases, this can result in programs aborting and segmentation violations. The cure is to ensure that the called and calling routines return values of compatible length and type.

An as interim fix, an environment variable can be set to cause the 4GL runtime system to pass parameters the way 6.4 did: all integers forced to 4-byte, all floats forced to 8-byte. Character string passing is not affected. The environment variable setting is:

```
setenv II_PARAM_PASSING FORCEMAX
```

Journaling On by Default

In Ingres 6.4, if a database was journaled, a newly-created table would not be journaled unless `WITH JOURNALING` was explicitly stated.

In Ingres, journaling is on by default. This means that if an application creates temporary tables, those tables will be journaled; this may consume more system resource, resulting in Ingres applications running more slowly than expected.

You can turn default journaling off by changing the Configuration-By-Forms parameter “default_journaling.” Alternative options are to issue a `SET NOJOURNALING` statement at the beginning of an application, create temporary tables `WITH NOJOURNALING`, or use session tables.

Greater Sensitivity to Arithmetic Errors

Ingres 6.4 ignores a number of arithmetic error conditions (such as floating point overflow and divide-by-zero). Ingres correctly reports arithmetic errors on all platforms. If an application generates arithmetic exceptions when tested with Ingres, it is probable that the application had problems in Ingres 6.4 that were not reported. The application must be corrected.

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 ))
```

Some 6.4 releases of 4GL had problems with variables of type `TABLE_KEY`. If you were doing type conversions to avoid the use of `TABLE_KEY` variables, consider removing the conversion altogether and using the `TABLE_KEY` type directly.

User-Defined Data Type Changes

If you are using Object Management Extension to declare user-defined data types in the server, be aware of some changes in calling sequences. For details, see the *Object Management Extension User Guide*.

Summary

Many of the changes required for Ingres are backward compatible with Ingres 6.4. Make application changes in the Ingres 6.4 installation, and bring them forward to the Ingres installation for testing. In this way, you do not have to freeze application development while preparing for Ingres.

At this stage, resist the temptation to make Ingres-specific application changes. While an outer join or a session temp table may enhance performance, there is plenty of time to add performance enhancements **after** the upgrade.

Preparing Your System

Take the following system preparation steps.

Ingres Startup and Shutdown

Ingres uses new commands for startup and shutdown: `ingstart` and `ingstop` instead of `iistartup` and `iishutdown`. If you have customized shell scripts that start and stop Ingres, you must change them. Verify the changes in the development Ingres installation and have the revised scripts ready for the production environment at time of upgrade.

If you are running multiple DBMS servers with Ingres 6.4, you should be able to simplify your startup and shutdown procedures. Ingres supports multiple DBMS servers directly from the Ingres configuration.

`ingprenv` Replaces `ingprenv1`

In Ingres, the `ingprenv` command replaces the Ingres 6.4 `ingprenv1` command, which displayed one Ingres environmental variable. Shell scripts that use `ingprenv1` must be changed.

It is possible to recreate `ingprenv1` as follows:

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

Archiver Exit Shellscript

Ingres has a sample Archiver exit script, `acpexit.def`. If the Ingres 6.4 `acpexit` script was customized, you must carry over these changes to the Ingres installation.

For information about the `acpexit` script, see the chapter “Customization Options” in the *System Administrator Guide*.

Transaction Log Size

Generally, Ingres uses less transaction log file space than Ingres 6.4. A few operations may use more (for example, `MODIFY TO MERGE`). To allow for its improved logging algorithms, Ingres reserves transaction log space that it may not actually write.

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 barely large enough, it may be advisable to increase the size before or during the upgrade.

Upgrading from 6.4 Using Upgradedb

Upgrading using `upgradedb` transforms your database in-place from Ingres 6.4 to the new version of Ingres.

Due to the large number of enhancements made, the `upgradedb` utility performs an intricate task when upgrading a 6.4 database. `Upgradedb` has been carefully tested, however, and most sites should be able to use it. In addition, the `upgradedb` procedure described in this section is designed so that the `upgradedb` utility has to perform as little work as possible, so that the utility will correctly handle the upgrade tasks.

In the procedure in this section, each database is prepared by dropping all objects that can be recreated, that is, by dropping everything but the base tables. Each base table must be checked to make sure it is valid and has no internal damage. After the upgrade, the various database objects are recreated.

The procedure directs you to cut and paste the output of `unloaddb` to generate SQL that recreates database objects and storage structures. If procedures already exist to recreate database objects and storage structures, you can use these instead. Make sure, however, that the procedures recreate all the relevant objects. If users or applications dynamically create database objects, it may be safer to cut and paste from `unloaddb`.

The `upgradedb` procedure assumes that you can become any user who owns objects in any database (using `login` or UNIX “`su`”). If this is not feasible, you can run as the installation owner (default user ID is `ingres`), and use the `-u{user}` flag to pretend to be that user whenever you must run an Ingres command.

Procedure

Follow this step-by-step procedure to perform an upgradedb upgrade from Ingres 6.4.

Procedure Notation In this procedure, the notation **[Each DB]** means: “For each database, not including the iidbdb, become the DBA for that database, **cd** to the unload directory for the database created in Step 1, and perform this step.”

Do not include the iidbdb or Ingres Distributed Option databases unless instructed.

If using the Distributed Option, remember to include the coordinator database in the list of databases.

Step 1: [Each DB Including Distributed Option DDBs] Create Unload Directory

Create a directory for each database. This directory is used to hold various scripts (but no data). The disk space needed is a maximum of 1 MB per directory. Make the directory writable by anyone.

UNIX

On UNIX, use these commands:

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

Windows

On Windows, use this command:

```
mkdir d:\someplace\dbname
```

Step 2: [Each DB Including Distributed Option DDBs] Run Unloaddb

Note on Steps 2 through 4: You can omit Steps 2 through 4 if procedures already exist to recreate all database objects and storage structures. However, it will be necessary to make the appropriate changes to the oi_prep.sh script (see [Step 9: \[Each DB\] Remove Non-table Objects](#) in this appendix) for re-modifying all tables.

Run unloaddb against each database. The unloaddb command does not unload the database; it simply creates copy in and copy out scripts. You can edit these scripts to produce a collection of scripts that recreate various database objects and storage structures.

Note: If using the Distributed Option, unload the CDB in the same way as for a local database. For a DDB, use unloaddb with the /star option.

For a regular DB or CDB:

```
unloaddb dbname
```

For a Distributed Option DDB:

```
unloaddb dbname/star
```

Step 3: [Each DB Including Distributed Option DDBs] Check for Obsolete Users

Examine the unload.ing and reload.ing scripts created in Step 1. Each script contains one line for each user who owns a database object. Make sure that all the users listed are valid; old databases may have objects created by users who no longer exist.

For any unwanted users, delete the relevant lines from unload.ing and reload.ing. Delete the cp{user}.in and cp{user}.out files, and go into the database and clean out the unwanted objects.

Step 4: [Each DB] Edit the Unloaddb Output

The unloaddb output needs to be modified for recreating just the database objects and storage structures. You must edit each cp{user}.in file that unloaddb created to extract the following statements:

- Create rule statements into a file named {user}_rule.sql
- Create procedure related statements into {user}_dbp.sql
- Create dbevent related statements into {user}_event.sql
- Modify statements into {user}_modify.sql
- Modify and create index statements into {user}_modindex.sql
- All other non-base-table related statements into {user}_grantview.sql. This file will contain grants, QUEL permits, QUEL integrities, and view definitions.

You can perform this step manually with a text editor.

For UNIX, the extract_unloaddb.sh shellscript is available that extracts one user's object definitions. The script is available on the Computer Associates Technical Support web site, accessible from <http://ca.com>.

The \$ingres user should not own any non-catalog objects, so do not process the cp_ingre.in file that unloaddb creates.

As a result of this step, SQL scripts have been created that can recreate any database object or storage structure owned by any user in any database.

Step 5: [Each DB Including iiddb] Checkpoint the Database (Optional)

Note: This step is optional because you will take a system backup and another checkpoint later.

Checkpoint all databases. Copy the checkpoint files to tape and verify that the tape can be read.

Step 6: Disable User Access

Disable user access.

From this step through the end of the upgrade, the production system is not available for use. Make sure that users are not able to access the databases until the upgrade is complete.

Step 7: Shut Down Ingres and Back Up System

Perform a clean shutdown of Ingres, clearing all transactions from the transaction log. To achieve this, shut down Ingres, restart Ingres, and then shut it down again. Check the recovery process log (II_RCP.LOG) for the message "RCP Shutdown completed normally."

Take a system backup using a command appropriate to the platform. Make sure you include all Ingres directories: data, checkpoint, journal, dump areas, and the \$II_SYSTEM/ingres directory containing Ingres files and executables. Back up the application directories.

Watch for symbolic links and cross-mounts; make sure real data is saved and not a symbolic link.

If Ingres is typically started up at boot time, include the root file system in the backup. Alternatively, print a copy of any Ingres boot time startup and shutdown scripts.

For maximum safety, perform this step twice. At minimum, after the backup completes, check the backup media to ensure that the backup can be read. If your backup medium is tape, use new tapes, and clean the tape drive before the backup.

After completing the backup, restart Ingres.

Step 8: [Each DB] Print Optimizer Statistics (Optional)

If your upgrade plan allows enough downtime to run a full `optimizedb` against your databases (after the upgrade), you can omit this step. If your plan does not allow enough downtime, perform this step as a shortcut.

Note: Be aware that using this shortcut may result in some of the new Ingres metrics not being available; query performance may suffer until a full `optimizedb` can be completed.

Dump the existing optimizer statistics. Run `statdump` with the `-o` flag to a file for each database, as follows:

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

Step 9: [Each DB] Remove Non-table Objects

Drop all non-table objects from the database including:

- Optimizer statistics
- Views
- Rules
- Database procedures
- Database events
- Secondary indexes
- Grants and QUEL permits
- QUEL integrities

In addition, modify all tables to heap.

The purpose of this step is to reduce the database to base tables.

Some database objects such as procedures and views can be very complicated, and some past versions of `upgradedb` did not always process them successfully.

Additionally, processing of some objects (grants in particular) is slow and expensive. Dropping the grants and later recreating them avoids any possible failure due to lack of transaction log space.

`oi_prep.sh`

To perform this task automatically, you can use the shell script `oi_prep.sh`. The script is available at the Computer Associates Technical Support web site, accessible from <http://ca.com>.

Using the C shell:

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

If there are any dependent views, “drop” errors messages may be reported on those views; (oi_prep.sh does not bother to drop views in reverse dependency order). Ignore those “drop” errors.

Verifydb

Run verifydb checks against the database. The verifydb -odbms command may output messages:

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 can ignore these messages. Also, ignore the “patch warning” message that warns of the loss of user tables in “runinteractive” mode. This mode will not be used.

Some databases may produce a “verifydb failed” message and then abort. If this happens, run the Terminal Monitor with the update system catalogs flag, as follows:

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

No rows should be returned. If there are rows, this is the probable cause of the verifydb problem. Delete the rows:

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

Rerun the verifydb command as shown at the end of the oi_prep.sh. If error messages are returned from verifydb, correct the problems before continuing. Contact Computer Associates Technical Support for help if necessary.

Do not process Distributed Option distributed databases.

Step 10: [Each DB] Record Database Information

Run infodb against each database, saving the output. The output will be needed later. You will need to know, for example, whether the database was journaled, the data locations where the database resides, and in what order the locations were configured.

Run infodb as follows:

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

If no dbname is specified, infodb prints a report for each database.

Step 11: Clean iidbdb Database

Become the user that owns the Ingres installation, and run a subset of Step 9 (see [Step 9: \[Each DB\] Remove Non-table Objects](#) in this appendix) against the master database iidbdb. It is assumed that there are no objects created by users in the iidbdb.

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

Messages from verifydb can be handled in the same way as in Step 9.

Step 12: [Each DB Including iidbdb] Checkpoint and Turn Off Journaling

Checkpoint each database, using the ckpdb command with -j option to turn off journaling.

If upgradedb fails, you can use this checkpoint to recover and try again.

Save the configuration file stored in the dump area after each checkpoint. The configuration file is small.

To do this, issue these commands:

```
ckpdb -d -j dbname
cp $II_DUMP/ingres/dmp/default/dbname/aaaaaaaa.cnf {somewhere secure}
```

Tip: The iidbdb needs the -s option for ckpdb. The iidbdb database does not have an “unload” directory. Store the aaaaaaaaa.cnf file in a safe place.

Step 13: Record Ingres Configuration

As the installation owner, execute the “showrcp” command and record the results.

Record the contents of the rundbms.opt file found in \$II_SYSTEM/ingres/files.

You will use this information later as a guide for configuring Ingres. The Ingres installation procedure does not preserve the Version 6.4 parameter settings. During installation, the ingres/files directory is deleted, so save the information.

Step 14: Shut Down Ingres

Shut down Ingres with the `iishutdown` command.

Step 15: Disable Ingres Startup

If Ingres starts automatically when the machine boots up, turn auto-starting off until the upgrade is complete.

On most UNIX platforms, a file in a system startup directory performs Ingres startup and shutdown; place an “exit 0” at the top of this file. The system administrator may need to perform this step if it requires root privilege. (The system startup directory depends on your platform — `/etc/init.d`, or `/sbin/init.d`, or a similar name).

On Windows, if Ingres is run as a system service, set the service to start manually instead of automatically.

Make sure that the operating system is correctly configured for your new version of Ingres (see [Preparing Your System](#) in the chapter “Getting Started”).

Reboot, if necessary, to put the operating system parameter changes into effect.

Step 16: Preserve Site Modifications

If you have customized any files that are distributed as part of Ingres, you must copy them because they will be lost during the upgrade. Any custom files you have **added** to the `$II_SYSTEM` directory tree will remain.

Commonly, customizations are made to the termcap and keyboard map files in `$II_SYSTEM/ingres/files`. Customizations are also made to local collation sequence files. Save the original collation definition files and the compiled files in `$II_SYSTEM/ingres/files/collation`.

Copy customized files to a safe place. Do **not** copy them to `/tmp` or anywhere in `$II_SYSTEM/ingres` directory.

If you cannot identify all your customized files, you can do the following to ensure that you preserve the necessary files:

1. Delete all `.log` and `.LOG` files from `$II_SYSTEM/ingres/files`
2. Copy to a safe place the entire contents of the following directories:
 - all `.opt` files
 - `$II_SYSTEM/ingres/bin`

- \$II_SYSTEM/ingres/files
- \$II_SYSTEM/ingres/rep
- \$II_SYSTEM/ingres/files/rep
- \$II_SYSTEM/ingres/files/dayfile
- \$II_ingres/files/startup
- \$II_SYSEM/ingres/files/startsql
- \$II_SYSTEM/ingres/utility

This procedure copies more files than necessary, but you can delete the copy after Ingres has been running live for a period.

Do not delete the copy immediately when the upgrade completes, because you may discover weeks later that you need the old version of a file (for example, a Vision template or keyboard map) from the original \$II_SYSTEM/ingres directory.

UNIX

On UNIX, to copy these files, use commands similar to the following:

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

Step 17: Fix Logins

If necessary, make sure that the login for the installation owner sets LD_LIBRARY_PATH or the platform equivalent. Make sure that the login for the user does not use ingpreenv1, or install your ingpreenv1 substitute. See [Preparing Your System](#) in the chapter “Getting Started.” Check all your database owner (DBA) logins to ensure that they are properly set up for Ingres, with LD_LIBRARY_PATH or equivalent, and no use of ingpreenv1.

LD_LIBRARY_PATH or equivalent must be defined for the user session that you will use to install and upgrade Ingres.

Step 18: Save Ingres Settings

The upgrade runs more smoothly if the Ingres 6.4 executables, control files, and environment variables are deleted, which you will do in the next step. However, you do not want to lose your installation ID and default locations, which are kept in a file named symbol.tbl.

Copy \$II_SYSTEM/ingres/files/symbol.tbl to a safe area not in the Ingres directory tree.

Step 19: Clean Up Ingres 6.4

To guarantee a clean environment for Ingres, invoke the following commands. The `rm` command removes existing 6.4 files.

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

Copy your saved `symbol.tbl` back into the `$II_SYSTEM/ingres/files` directory.

Note: Issuing the above command will cause Ingres Net definitions to be lost. As an alternative, you can run the `rm` command without specifying the files subdirectory. The Ingres Net definitions will be preserved, but the installation will not be as clean.

Step 20: Create Work Location

The Ingres installation procedure asks for a location for temporary files and sorting, and creates the directories if they do not exist. However, you should create this location manually because some versions of the installation procedure may not properly set the protections for the directories, which can cause `upgradedb` to fail when upgrading the `iidbdb` database.

For information on placement of your default work location, see the *Database Administrator Guide*.

As the installation owner, assume a work location called `/mywork`:

UNIX

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

Windows

```
md \mywork\ingres
md \mywork\ingres\work
md \mywork\ingres\work\default
md \mywork\ingres\work\default\iidbdb
```

Step 21: Install Ingres

To install Ingres, see the Ingres installation instructions for your platform.

During the installation process, the DBMS Server setup asks whether all databases are to be upgraded; answer **No**. The installation procedure automatically upgrades the iidbdb. If the upgrade of iidbdb fails, see the appendix “[Troubleshooting Upgradedb Problems](#).” It is better to complete the Ingres setup, and then use the upgradedb command to upgrade the user databases.


After the iidbdb is upgraded, the DBMS Server setup attempts to upgrade imadb and install Remote Command Server objects into imadb. Some versions of upgradedb neglect to create imadb first, and you will get “Database does not exist: imadb” errors. These will be corrected in the next step.

Upgrading to Versions That Require a Patch

UNIX

You can install Ingres service packs without having to install a base release of Ingres first. If you are upgrading to an older Ingres version that requires an overlay patch instead of a service pack, follow this procedure:

1. Run ingbuild. When asked whether you want to set up all the Ingres components, respond **No**. Exit ingbuild.
2. Install the Ingres patch.
3. Run ingbuild again. Select Current, then SetupAll.
4. Follow the prompts to complete the Ingres setup.

Setup now uses the fixed version. 


Step 22: Create imadb Database

Note: Perform this step only if you received “Database does not exist: imadb” messages during the DBMS setup phase of your Ingres install.

Create the imadb database.

As the installation owner, execute these commands:

UNIX

```
ingstart
cd $II_SYSTEM/ingres/vdba
createdb '-u$ingres' imadb -f nofeclients
sql '-u$ingres' imadb <makimau.sql
rmcmdgen
ingstop 
```

Windows

```
ingstart
cd %II_SYSTEM%\ingres\vdbs
createdb -u$ingres imadb -f nofeclients
sql -u$ingres imadb <makiman.sql
rmcmdgen
ingstop
```

As the makimau or makiman SQL scripts run, you see a series of messages such as “E_US0AC1 'some-name' does not exist or is not owned by you.” These are normal and can be ignored.

Step 23: Restore Site Modifications

Restore any site-specific files that you copied in Step 16 (see [Step 16: Preserve Site Modifications](#) in this appendix).

- | | |
|----------------------|---|
| Checkpoint Template | If the checkpoint template file cktmpl.def has been modified, the modifications may need to be carried forward into Ingres. The cktmpl.def from Ingres 6.4 cannot be used with Ingres, as the file format has changed. This means that you must recreate the changes using the Ingres 6.4 cktmpl.def as a guide. See the <i>Ingres 6.4 Database Administrator's Guide</i> . |
| Archiver Exit Script | If the archiver exit script acpexit was changed in Ingres 6.4, you must make the changes to the Ingres template (acpexit.def), and then move that file to \$II_SYSTEM/ingres/files/acpexit. |

Step 24: Start Ingres

Run ingstart to start Ingres.

Step 25: Run Upgradedb Utility

Run the upgradedb utility to upgrade databases.

You can upgrade databases one at a time or all at the same time. Log the upgradedb output to a file.

To upgrade one at a time:

```
upgradedb dbname
```

To upgrade all at the same time:

```
upgradedb -all
```


Example of logging upgradedb output to a file:

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

Troubleshooting

If errors occur, see the appendix “[Troubleshooting Upgradedb Problems](#).” Correct the errors and rerun the upgradedb utility.

If the upgradedb command starts and then hangs with no error indication, the Remote Command Server may be interfering with the process. This is particularly likely if you are upgrading to Ingres II version 2.0 instead of Ingres. To fix, stop the Remote Command Server, as follows:

```
rmcmdstp
```

You can use Configuration-By-Forms or Visual Configurator to turn off the Remote Command Server until your upgrade is finished: select Remote Command Server and use EditCount to set the startup count to zero.

Step 26: Configure Ingres

Run Configuration-By-Forms (CBF) and initially configure the Ingres installation. Use the rundbms.opt and showrcp information from Ingres 6.4 as a guideline. For information about CBF and the various tuning parameters, see the *System Administrator Guide*.

For information on the correlation between 6.4 and Ingres parameter names, see [Corresponding Parameter Names](#) in this appendix.

Derived parameters are recalculated when values they depend on are changed. If derived parameters are set, they can be “protected” against change.

Ingres versions from 2.0 through 2.6 may calculate very large default lock and resource limits parameters. Check the lock_limit and resource_limit settings, and consider reducing these limits to the Ingres 6.4 settings.

On OS-thread platforms, do not turn on async_io; and do not declare the II_NUM_SLAVES Ingres variable.

Ingres supports larger qef_sort_mem values than Ingres 6.4. Ingres may not need as much qsf_memory as did Ingres 6.4. OS-thread platforms should not reduce quantum_size, as it does not improve performance on those platforms.

Step 27: Set Up Ingres Net

Run netutil to create the vnode definitions for the remote installations. If installation passwords are needed, you must run mkvalidpw. See the *System Administrator Guide* or the Readme for your platform.

If there are NFS client-only installations that have not been set up, run `ingmknfs` to set them up.

Step 28: [Each DB] Recreate Objects

Using the scripts generated by Step 4 (see [Step 4: \[Each DB\] Edit the Unloaddb Output](#) in this appendix), recreate the views.

Recreate in the following sequence:

1. Views, QUEL integrities, and grants:
`sql -uuser dbname <user_grantview.sql`
2. Dbevents:
`sql -uuser dbname <user_event.sql`
3. Database procedures:
`sql -uuser dbname <user_dbasp.sql`
4. Rules:
`sql -uuser dbname <user_rule.sql`

Remember to run all four scripts for each user who owns objects in each database.

If your application system has its own scripts to recreate database objects, you may use them instead of the unloaddb-generated scripts.

Step 29: [Each DB] Reapply Storage Structures

For each `user_modindex.sql` script generated by Step 4 (Editing the Unloaddb Output), reapply storage structures and indexes:

```
sql -uuser dbname <user_modindex.sql
```

If your application system has its own scripts to reapply storage structures and create indexes, you may use them instead of the unloaddb-generated scripts.

Step 30: [Each DB] Reapply Optimizer Statistics

Regenerate optimization statistics. You can do this either by regenerating statistics from scratch or by using the original statistics printed from the Ingres 6.4 installation earlier in this upgrade procedure (see [Step 8: \[Each DB\] Print Optimizer Statistics \(Optional\)](#) in this appendix).

If there is sufficient time, we recommend that you regenerate the optimizer statistics using the procedures of your application system. Ingres computes more statistics than did 6.4.

If time is short, and if you printed the original statistics in Step 8, you can read them back in with the `-i` option to `optimizedb`:

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

Step 31: [Each DB including iidbdb] Checkpoint the Database

Checkpoint each database. If the database was journaled previously, use the `+j` flag to turn on journaling.

To know which databases were journaled, see the `infodb` output from Step 10 (see [Step 10: \[Each DB\] Record Database Information](#) in this appendix).

The `iidbdb` should always be journaled, regardless of whether it was journaled in the 6.4 installation.

Step 32: Install Upgraded Applications

Install the Ingres versions of the applications. Then restore user logins and resume normal operation.

This completes the `upgradedb` upgrade procedure.

Upgrading from 6.4 Using Unload/Reload

The unload/reload upgrade avoids the `upgradedb` program (except for `iidbdb`), in favor of unloading the Ingres 6.4 databases to flat files, recreating the databases under Ingres, and then reloading the databases. This approach has the advantage of starting with clean databases, but requires more time and disk space than does the `upgradedb` method.

Note: Databases using the system-maintained logical key feature are best upgraded using `upgradedb`. Tables that contain `SYSTEM_MAINTAINED` `table_key` or `object_key` columns cannot be safely unloaded and reloaded without additional work. The reload step generates all new logical key values. If other tables reference the logical key columns, the new values must be manually propagated to those tables.

Two Upgrade Types

You must choose one of the following variations of the unload/reload procedure:

- In-place upgrade, which replaces the 6.4 installation with the new Ingres installation. The master database (iiddb) is upgraded with upgradedb, even though other databases are unloaded and reloaded. Because the iiddb remains, all your locations, users, groups, and roles still exist in the new installation.
- Clean install upgrade, which leaves the 6.4 installation alone. Ingres is installed into a completely new installation. (The new installation may even be on a different machine.) When performing a clean install upgrade, you must take extra steps to recreate locations and move users, groups, and roles from the 6.4 installation to the new one.

Front-end Catalogs

The hardest part of the unload/reload upgrade is dealing with the front-end catalogs. These catalogs are unloaded in Ingres 6.4 format, and cannot be loaded into an Ingres database. To circumvent this problem, the Ingres database is created without front-end catalogs. The catalogs are then loaded in the Ingres 6.4 format and upgraded using the upgrade program.

Procedure

Follow this step-by-step procedure to perform an unload/reload upgrade from Ingres 6.4.

Procedure Notation

In this procedure, the notation **[Each DB]** means: “For each database, not including the iiddb, become the DBA for that database, **cd** to the unload directory for the database created in Step 1, and perform this step.”

If using the Distributed Option, include the coordinator database in the list of databases.

Steps that apply to a particular upgrade type only (that is, in-place upgrade or clean install upgrade) are marked accordingly.

Step 1: [Each DB Including iiddb] Create Unload Directory

Create a directory for each database. The directory is used to hold scripts and data from the unloaded database and requires a large amount of disk space. As an estimate, the unloaded data is about the same size as the Ingres database; however, compressed data can expand to take up much more space than the Ingres database.

UNIX

On UNIX, use these commands:

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

Windows

On Windows, use this command:

```
mkdir d:\someplace\dbname
```

Step 2: [Each DB] Run Unloaddb

Run unloaddb against each database. The unloaddb command does not unload the database; it simply creates scripts.

Note: If using the Distributed Option, unload the CDB in the same way as for a local database. For a DDB, use unloaddb/star.

For a regular DB or CDB:

```
unloaddb dbname
```

For a Distributed Option DDB:

```
unloaddb dbname/star
```

If doing a clean-install upgrade to a different machine that has a newer architecture, binary data may not be compatible between the two machines. If this is the case, use the unloaddb **-c** option, which causes an ASCII instead of binary unload.

Step 3: [Each DB] Check for Obsolete Users

Old databases may have objects created by users who no longer exist.

Examine the unload.ing and reload.ing scripts created in Step 1. Each script contains one line for each user who owns a database object. Make sure that all the users listed are valid.

For any unwanted users, delete the relevant lines from the unload.ing and reload.ing scripts. Delete the cp{user}.in and cp{user}.out files, and go into the database and clean out these unwanted objects.

Step 4: [Each DB Including iidbdb] Checkpoint the Database (Optional)

Note: This step is optional. You can omit this step if you can rely on the system backup to be taken in Step 6 (see [Step 6: Shut Down Ingres and Back Up System](#) in this appendix).

Checkpoint each database and then copy the checkpoint files to a permanent medium such as tape. If using tape, use fresh tape and verify that the tape can be read.

Remember to checkpoint the iidbdb.

Step 5: Disable User Access

Disable user access.

Step 6: Shut Down Ingres and Back Up System

Note: This step is required only for an in-place upgrade. However, this is a convenient time to get a complete backup of your production system before the upgrade. Therefore, you may want to perform this step even if doing a clean install upgrade.

Perform a clean shutdown of Ingres, clearing all transactions from the transaction log. To achieve this, shut down Ingres, restart Ingres, and then shut it down again. Check the recovery process log (II_RCP.LOG) for the message "RCP Shutdown completed normally."

Take a system backup using a command appropriate to the platform. Make sure you include all Ingres directories: data, checkpoint, journal, dump areas, and the \$II_SYSTEM/ingres directory containing Ingres files and executables. Back up the application directories.

Watch for symbolic links and cross-mounts; make sure real data is saved and not a symbolic link.

If Ingres is typically started up at boot time, include the root file system in the backup. Alternatively, print a copy of any Ingres boot time startup and shutdown scripts.

For maximum safety, perform this step twice. At minimum, after the backup completes, check the backup media to ensure that the backup can be read. If your backup medium is tape, use new tapes, and clean the tape drive before the backup.

After completing the backup, restart Ingres.

Step 7: [Each DB] Unload the Database

For each database, run the unload.ing script created by unloaddb. This unloads the database into your unload directory.

Step 8: [Each DB] Print Optimizer Statistics (Optional)

If your upgrade plan allows enough downtime to run a full optimizedb against your databases, you can omit this step. If your plan does not allow enough downtime, perform this step as a shortcut.

Note: Be aware that using this shortcut may result in some of the new Ingres metrics not being available; query performance may suffer until a full optimizedb can be completed.

Dump the existing optimizer statistics. Run statdump with the -o flag to a file for each database, as follows:

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

Step 9: [Each DB] Record Database Information

Run infodb against each database, saving the output. The output will be needed later. You will need to know, for example, whether the database was journaled, the data locations where the database resides, and in what order the locations were configured.

Run infodb as follows:

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

Also, record whether the database is public or private. To find out, use the catalogdb command. Select Databases, and then enter the database name. The screen that appears has an Access field that indicates whether the database is public or private.

Step 10: Record Database Privileges

As the installation owner, change directories to the unload directory for iidbdb created in Step 1. Save private database access lists and user database privileges, as follows:

```
sql iidbdb
\script dbaccess.out
select dbname, username
from iidbaccess
order by dbname, username
\go
\script
\script dbprivs.out
select *
from iidbprivileges
where database_name <> ''
order by database_name, grantee_name
\go
\script
\quit
```

This procedure creates two files, dbaccess.out and dbprivs.out.

Step 11: Save Users, Groups, and Roles

Note: This step is required only for a clean-install upgrade.

As the installation owner, change directory to the iidbdb unload directory, as you did in the previous step. Run the following SQL to save users, groups, and roles:

```
sql iidbdb
create table unload_tmp as
select name, status, default_group
from iiuser
where name not in ('ingres', '$ingres', 'root')
\go
copy unload_tmp (
    name=c0comma, status=c0comma, default_group=c0nl
) into 'users.out'
\go
drop unload_tmp; commit
\go

copy iiusergroup (
    groupid=c0comma, groupmem=c0nl
) into 'groups.out'
\go

copy iirole(
    roleid=c0nl
) into 'roles.out'
\go
\quit
```


Step 12: [Each DB] Destroy the Database

Note: This step is required only for an in-place upgrade.

Destroy each database using the `destroydb` command.

Step 13: Clean iidbdb Database

Note: This step is required only for an in-place upgrade.

Become the installation owner, and run the following steps against the master database iidbdb. It is assumed that there are no objects created by users in the iidbdb.

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

The `verifydb -odbms` command may issue the following messages:

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 can ignore these messages. Also, ignore the “patch warning” message that warns of the loss of user tables in “runinteractive” mode. This mode will not be used.

Step 14: Record Ingres Configuration

As the installation owner, execute the “showrcp” command and record the results.

Record the contents of the `rundbms.opt` file found in `$IIL_SYSTEM/ingres/files`.

You will use this information later as a guide for configuring Ingres. The Ingres installation procedure does not preserve the Version 6.4 parameter settings. During installation, the `ingres/files` directory is deleted, so save the information.

Step 15: Shut Down Ingres

Shut down Ingres with the `iishutdown` command.

Step 16: Disable Ingres Startup

Note: This step is recommended even if you are doing a clean installation upgrade. By leaving the old 6.4 installation shut down, you eliminate the chance that someone will connect to it by mistake later.

If Ingres starts automatically when the machine boots up, turn auto-starting off until the upgrade is complete.

On most UNIX platforms, a file in a system startup directory performs Ingres startup and shutdown; place an “exit 0” at the top of this file. The system administrator may need to perform this step if it requires root privilege. (The system startup directory depends on your platform — `/etc/init.d`, or `/sbin/init.d`, or a similar name).

On Windows, if Ingres is run as a system service, set the service to start manually instead of automatically.

Make sure that the operating system is correctly configured for your new version of Ingres (see [Preparing Your System](#) in the chapter “Getting Started”).

Reboot, if necessary, to put the operating system parameter changes into effect.

Step 17: Preserve Site Modifications

If you have customized any files that are distributed as part of Ingres, you must copy them because they will be lost during the upgrade. Any custom files you have **added** to the `$II_SYSTEM` directory tree will remain.

Commonly, customizations are made to the termcap and keyboard map files in `$II_SYSTEM/ingres/files`. Customizations are also made to local collation sequence files. Save the original collation definition files and the compiled files in `$II_SYSTEM/ingres/files/collation`.

Copy customized files to a safe place. Do **not** copy them to `/tmp` or anywhere in `$II_SYSTEM/ingres` directory.

If you cannot identify all your customized files, you can do the following to ensure that you preserve the necessary files:

1. Delete all `.log` and `.LOG` files from `$II_SYSTEM/ingres/files`

2. Copy to a safe place the entire contents of the following directories:

- all .opt files
- \$II_SYSTEM/ingres/bin
- \$II_SYSTEM/ingres/files
- \$II_SYSTEM/ingres/rep
- \$II_SYSTEM/ingres/files/rep
- \$II_SYSTEM/ingres/files/dayfile
- \$II_SYSTEM/ingres/files/startup
- \$II_SYSTEM/ingres/files/startsql
- \$II_SYSTEM/ingres/utility

This procedure copies more files than necessary, but you can delete the copy after Ingres has been running live for a period.

Do not delete the copy immediately when the upgrade completes, because you may discover weeks later that you need the old version of a file (for example, a Vision template or keyboard map) from the original \$II_SYSTEM/ingres directory.

UNIX

On UNIX, to copy these files, use commands similar to the following:

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

Step 18: Fix Logins

If necessary, make sure that the login for the installation owner sets LD_LIBRARY_PATH or the platform equivalent. Make sure that the login for the user does not use ingpreenv1, or install your ingpreenv1 substitute. See [Preparing Your System](#) in the chapter “Getting Started.” Check all your database owner (DBA) logins to ensure that they are properly set up for Ingres, with LD_LIBRARY_PATH or equivalent, and no use of ingpreenv1.

LD_LIBRARY_PATH or equivalent must be defined for the installation owner user session that you will use to install and upgrade Ingres.

If you are doing a clean-install upgrade on a different machine, make sure that your login fixes are applied to the new machine, not to the old one.

Step 19: Save Ingres Settings

Note: This step is required only for an in-place upgrade.

The upgrade runs more smoothly if the Ingres 6.4 executables, control files, and environment variables are deleted. However, you do not want to lose your installation ID and default locations. These are kept in a file named `symbol.tbl`.

Copy `$II_SYSTEM/ingres/files/symbol.tbl` to a safe area not in the Ingres directory tree.

Step 20: Clean Up Ingres 6.4

Note: This step is required only for an in-place upgrade.

To guarantee a clean environment for Ingres, invoke the following commands:

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

Copy your saved `symbol.tbl` back into the `$II_SYSTEM/ingres/files` directory.

Step 21: Create Work Location

Note: This step is required only for an in-place upgrade.

The Ingres installation procedure asks for a location for temporary files and sorting, and creates the directories if they do not exist. However, you should create this location manually because some versions of the installation procedure may not properly set the protections for the directories, which can cause `upgradedb` to fail when upgrading the `iidbdb` database.

For information on placement of your default work location, see the *Database Administrator Guide*.

As user `ingres`, assume a work location called `/mywork`:

UNIX

```
/mywork:
mkdir /mywork/ingres
mkdir /mywork/ingres/work
mkdir /mywork/ingres/work/default
mkdir /mywork/ingres/work/default/iidbdb
chmod 755 /mywork/ingres
chmod 700 /mywork/ingres/work
```

Windows

```

chmod 777 /mywork/ingres/work/default
chmod 777 /mywork/ingres/work/default/iidbdb
md \mywork\ingres
md \mywork\ingres\work
md \mywork\ingres\work\default
md \mywork\ingres\work\default\iidbdb

```

Step 22: Install Ingres

To install Ingres, see the Ingres installation instructions for your platform.


In-place upgrades only: During the installation process, the DBMS Server setup asks whether all databases are to be upgraded; answer **No**. The install procedure automatically upgrades the iidbdb. If the upgrade of iidbdb fails, see the appendix “[Troubleshooting Upgradedb Problems](#).”

After the iidbdb is upgraded, the DBMS Server setup attempts to upgrade imadb and install Remote Command Server objects into imadb. Some versions of upgradedb neglect to create imadb first, and you will get “Database does not exist: imadb” errors. These will be corrected in the next step.

Upgrading to Versions That Require a Patch**UNIX**

You can install Ingres service packs without having to install a base release of Ingres first. If you are upgrading to an older Ingres version that requires an overlay patch instead of a service pack, follow this procedure:

1. Run ingbuild. When asked whether you want to set up all the Ingres components, respond **No**. Exit ingbuild.
2. Install the Ingres patch.
3. Run ingbuild again. Select Current, then SetupAll.
4. Follow the prompts to complete the Ingres setup.

Setup now uses the fixed version. 

Step 23: Create imadb Database

Note: Perform this step only if you received “Database does not exist: imadb” messages during the DBMS Setup phase of your Ingres install.

Create the imadb database.

As the installation owner, execute these commands:

UNIX

```
ingstart
cd $II_SYSTEM/ingres/vdba
createdb '-u$ingres' imadb -f nofeclients
sql '-u$ingres' imadb <makimau.sql
rmcmdgen
ingstop
```

Windows

```
ingstart
cd %II_SYSTEM%\ingres\vdba
createdb -u$ingres imadb -f nofeclients
sql -u$ingres imadb <makiman.sql
rmcmdgen
ingstop
```

As the makimau or makiman SQL scripts run, you see a series of messages such as “E_US0AC1 'some-name' does not exist or is not owned by you.” These are normal and can be ignored.

Step 24: Restore Site Modifications

Restore any site-specific files that you copied in Step 17 (see [Step 17: Preserve Site Modifications](#) in this appendix).

- | | |
|----------------------|---|
| Checkpoint Template | If the checkpoint template file cktmpl.def has been modified, the modifications may need to be carried forward into Ingres. The cktmpl.def from Ingres 6.4 cannot be used with Ingres, as the file format has changed. This means that you must recreate the changes using the Ingres 6.4 cktmpl.def as a guide. See the <i>Ingres 6.4 Database Administrator's Guide</i> . |
| Archiver Exit Script | If the archiver exit script acpexit was changed in Ingres 6.4, you must make the changes to the Ingres template (acpexit.def), and then move that file to \$II_SYSTEM/ingres/files/acpexit. |

Step 25: Configure Ingres

Run Configuration-By-Forms (CBF) and initially configure the Ingres installation. Use the rundbms.opt and showrcp information from Ingres 6.4 as a guideline. For information about CBF and the various tuning parameters, see the *System Administrator Guide*.

For information on the correlation between 6.4 and Ingres parameter names, see [Corresponding Parameter Names](#) in this appendix.

Derived parameters are recalculated when values they depend on are changed. If derived parameters are set, they can be “protected” against change.

Ingres versions from 2.0 through 2.6 may calculate very large default lock and resource limits parameters. Check the `lock_limit` and `resource_limit` settings, and consider reducing these limits to the Ingres 6.4 settings.

On OS-thread platforms, do not turn on `async_io`; and do not declare the `II_NUM_SLAVES` Ingres variable.

Ingres supports larger `qef_sort_mem` values than Ingres 6.4. Ingres may not need as much `qsf_memory` as did Ingres 6.4. OS-thread platforms should not reduce `quantum_size`, as it does not improve performance on those platforms.

Step 26: Set Up Ingres Net

Run `netutil` to create the vnode definitions for the remote installations. If installation passwords are needed, you must run `mkvalidpw`. See the *System Administrator Guide* or the *Readme* for your platform.

If there are NFS client-only installations that have not been set up, run `ingmnfs` to set them up.

Step 27: Start Ingres

Run `ingstart` to start Ingres.

Step 28: Recreate Users, Groups, and Roles

Note: This step is required only for a clean-installation upgrade.

Users and Groups

If your 6.4 installation has only a few Ingres users defined, you should use the `accessdb` utility or the `CREATE USER SQL` statement to recreate those users in the Ingres installation. As a guide, use the file `users.out` or refer to the 6.4 installation.


If you have many users, the following procedure recreates them in mass.

As the installation owner, change directory to your `iidbdb` `unloaddb` directory where you stored the files from Step 11 (see [Step 11: Save Users, Groups, and Roles](#) in this appendix). Run this SQL:

```
sql '-u$ingres' iidbdb
copy iuser(name=c0comma,status=c0comma,default_group=c0nl)
from 'users.out'
\go
update iuser
set default_priv = status, user_priv = status,
    flags_mask = case when default_group <> ' ' then 28 else 24 end
where user_priv = 0 and flags_mask = 0;
```

```
\go
copy iusergroup(groupid=c0comma,groupmem=c0nl)
from 'groups.out'
\go
commit
\go
\quit
```

Windows

For Windows, omit the quotes from the sql command line. 

Ingres has new user privileges that do not exist in 6.4. If you recreate users using the above bulk load procedure, you should review the added users with `accessdb` to make sure that all user privileges are set the way you want them. In particular, review the definitions for any 6.4 “superusers.”

Ingres handles the “update system catalog” privilege differently than did 6.4. You must explicitly grant this privilege to the Ingres user after you recreate it, with a grant statement, as follows:

```
grant update_syscat on current installation to user-name
```

Roles

If your 6.4 installation had roles defined, recreate them with the ADD ROLE SQL statement. Use the file `roles.out` as a guide. Roles cannot be reliably bulk-loaded from the 6.4 installation, so you must recreate them by hand. After you recreate each role, issue the following SQL statement:

```
grant rolename to public; commit
```

This allows the role to be used in the same manner as in 6.4.

Step 29: Recreate Locations

Note: This step is required only for a clean-install upgrade.

Refer to **each** `infodb` output saved in Step 9 (see [Step 9: \[Each DB\] Record Database Information](#) in this appendix). Create any location that is not a default installation location (`ii_database`, `ii_checkpoint`, `ii_journal`, or `ii_dump`).

For more information about creating locations, see the *Database Administrator Guide*.

Step 30: [Each DB] Recreate the Database

Before creating each database, refer to the `infodb` output saved in Step 9 (see [Step 9: \[Each DB\] Record Database Information](#) in this appendix). Look at the location names for `ROOT`, `JOURNAL`, `CHECKPOINT`, and `DUMP`. If these are not `ii_database`, `ii_journal`, `ii_checkpoint`, or `ii_dump`, you must specify the location to `createdb` with the `-d`, `-j`, `-c`, or `-b` flags, respectively.

Also, refer to the database access information recorded in that step. If the database access was “private,” you must use the `-p` flag for `createdb`.

If all the database locations are the default, and the database is public, you can omit the flags on the `createdb` command line.

Recreate each user database, omitting the front-end catalogs. (The front-end catalogs will be reloaded and upgraded in a later step.) Use the following command:

```
createdb dbname flags -f nofeclients
```

Note: For a Distributed Option database, run `createdb/star` for the DDB. Do not run `createdb` for the CDB.

Step 31: [Each DB] Extend the Database

Refer to the `infodb` output saved in Step 9 (see [Step 9: \[Each DB\] Record Database Information](#) in this appendix). If the database was extended to data locations other than the default location, run `accessdb` as the installation owner and extend the newly-created databases to the same locations. The locations will already exist; it is only necessary to extend the databases to use them.

If you prefer a non-interactive command line utility, you can use the `extenddb` utility instead of `accessdb`.

Step 32: Recreate Database Privileges

As the installation owner, change to the `iidbdb unloaddb` directory, and refer to the file `dbaccess.out` created in Step 10 (see [Step 10: Record Database Privileges](#) in this appendix).

Start an `iidbdb` Terminal Monitor session:

```
sql iidbdb
```

For each database and user combination listed in `dbaccess.out`, issue the statement:

```
grant access on database database-name to username; commit
```

Next, review the file `dbprivs.out` created in Step 10. Each row describes one or more database privileges given to the user *grantee-name*. A Y or N in a privilege column indicates the specific privilege. (A U in a column means “Unchanged.”)

For each row, issue the statement

```
grant privilege on database database-name to grantee-name; commit
```

If the privilege column is N, grant *noprivilege* instead of *privilege*.

When finished, use **\quit** to exit the iidbdb session.

If you have defined many privileges, or recreated many users, groups, or roles, you should run **sysmod** on the iidbdb, which will accelerate query processing. Issue the **sysmod** command, as follows:

```
sysmod iidbdb
```

Step 33: [Each DB] Fix FE Reload Script

Edit the file `cp_ingre.in` and locate the lines:

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

Note: The directory path may differ.

Delete these lines and save the file. Because the database was not created with front-end catalogs, it is not necessary to drop them.

Step 34: [Each DB] Reload the Database

Run `reload.ing` for each database. Redirect the reload to a log file so that it can be checked for errors.

UNIX

Using the C shell:

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

Note: If using the Distributed Option, reload the CDB and all “real” local databases before reloading the DDBs. 📌

After the reload is complete, verify that the table `ii_id` has only one row. Type **isql** *<database>*, and **select * from ii_id**. If more than one row is returned, delete the row with the lowest `object_id`.

Step 35: [Each DB] Upgrade Front-end Catalogs

Run **upgradefe** on each database, which brings the front-end catalogs up to Ingres level. Issue the following command:

```
upgradefe dbname INGRES
```

The word INGRES should appear in uppercase.

Step 36: [Each DB] Reapply Optimizer Statistics

Regenerate optimization statistics. You can do this either by regenerating statistics from scratch or by using the original statistics printed from the Ingres 6.4 installation earlier in this upgrade procedure (see [Step 8: \[Each DB\] Print Optimizer Statistics \(Optional\)](#)).

If there is sufficient time, we recommend that you regenerate the optimizer statistics using the procedures of your application system. Ingres computes more statistics than did 6.4.

If time is short, and if you printed the original statistics in Step 8, you can read them back in with the `-i` option to `optimizedb`:

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

Step 37: [Each DB including iidbdb] Checkpoint the Database

Checkpoint each database. If the database was journaled previously, use the `+j` flag to turn on journaling.

To know which databases were journaled, see the `infodb` output from Step 9 (see [Step 9: \[Each DB\] Record Database Information](#) in this appendix).

The `iidbdb` should always be journaled, regardless of whether it was journaled in the 6.4 installation.

Step 38: Install Upgraded Applications

Install the Ingres versions of the applications. Then restore user logins and resume normal operation.

Corresponding Parameter Names

The configuration system in Ingres 6.4 differs from that of subsequent releases. This section discusses the Ingres 6.4 server parameters that correspond to Ingres parameters. All corresponding Ingres parameters listed are found in the DBMS Server component.

Ingres parameters that do not have any corresponding Ingres 6.4 parameters are not listed.

Parameters in 6.4 rundbms.opt File

Note: Parameters of type Cache are repeated for each cache page size.

Ingres 6.4 Parameter	Ingres Parameter	Type
active_sessions	active_limit	Derived
cache_name	cache_name	
connected_sessions	connect_limit	
cpu_statistics	cpu_statistics	Derived
cursors_per_session	cursor_limit	
database_count	database_limit	Derived
dblist	database_list	Databases
define	define_address	Derived
dmf.cache_size	dmf_cache_size	Cache, Derived
dmf.count_read_ahead	dmf_group_count	Cache, Derived
dmf.dbcache_size	dmf_db_cache_size	
dmf.flimit	dmf_free_limit	Cache, Derived
dmf.memory	dmf_memory	Cache, Derived
dmf.mlimit	dmf_modify_limit	Cache, Derived
dmf.scanfactor	dmf_scan_factor	Cache
dmf.size_read_ahead	dmf_group_size	Cache
dmf.tblcache_size	dmf_tbl_cache_size	
dmf.tcb_hash	dmf_hash_size	
dmf.wbend	dmf_wb_end	Cache, Derived
dmf.wbstart	dmf_wb_start	Cache, Derived
events	event_limit	
fast_commit	fast_commit	Derived
flatten	query_flattening (ON)	
image	image_name	
maximum_working_set	unix_maximum_working_set	
names	name_service (ON)	
noflatten	query_flattening (OFF)	

Ingres 6.4 Parameter	Ingres Parameter	Type
nonames	name_service (OFF)	
opf.active	opf_active_limit	Derived
opf.aggregate_flatten	qflatten_aggregate (ON)	Derived
opf.complete	opf_complete (ON)	
opf.cpubfactor	opf_cpu_factor	
opf.exactkey	opf_exact_key	
opf.memory	opf_memory	Derived
opf.noaggregate_flatten	qflatten_aggregate (ON)	Derived
opf.nocomplete	opf_complete (OFF)	
opf.nonkey	opf_non_key	
opf.rangekey	opf_range_key	
opf.repeatfactor	opf_repeat_factor	
opf.sortmax	opf_sort_max	
opf.timeoutfactor	opf_timeout_factor	
priority	unix_priority	
psf.memory	psf_memory	Derived
qef.qep_size	qef_qep_mem	
qef.sort_size	qef_sort_mem	
qsf.pool_size	qsf_memory	Derived
quantum	quantum_size	
rdf.max_tbls	rdf_max_tbls	
rdf.memory	rdf_memory	Derived
rdf.tbl_cols	rdf_tbl_cols	
rdf.tbl_idx	rdf_tbl_idx	
rule_depth	rule_depth	
scf.row_estimate	scf_rows	
server_class	server_class	
session_accounting	session_accounting	
shared_cache	cache_sharing (ON)	
sole_cache	cache_sharing (OFF)	

Ingres 6.4 Parameter	Ingres Parameter	Type
sole_server	sole_server	Derived
stack_size	stack_size	
write_behind	dmf_write_behind (see notes)	Cache

Notes on Specific Parameters

Note the following:

- The 6.4 QEF sorting algorithm is unsuited to large qef_sort_mem settings.
- All recent Ingres versions use a different sort that does not degrade with large qef_memory_settings. The 6.4 standard setting is much smaller than the Ingres default.
- Ingres 2.6 generally requires significantly less qsf_memory than Ingres 6.4 does, perhaps as little as half. After upgrading, start with the same qsf_memory setting as Ingres 6.4, but monitor QSF memory usage with trace point QS501 and tune qsf_memory appropriately.
- The quantum_size parameter in an internal threads (slaves) installation is often set to a small number (50 to 100) to improve responsiveness. Quantum_size has a different meaning in an OS threads installation, where it should not be set to less than 300, or excessive polling of the session communications channel will occur. A quantum_size of 1000 is usually appropriate when OS threads are in use.
- The 6.4 write_behind parameter is a thread count. Starting with 2.5, the dmf_write_behind parameter is simply ON or OFF, and the server dynamically allocates threads. Prior to Ingres II 2.5, the write_behind parameter means the same as it did in 6.4.
- A stack_size of 64 KB is typical with 6.4. Recent Ingres versions use more stack, so the stack size should be set to 128 KB (or more, if sporadic session failures occur).
- 6.4 VMS installations can have a few additional non-UNIX parameters, which have the same or almost the same names in Ingres 2.6.

Locking and Logging System Parameters

These parameters are set with iistartup -init, or rcpcnfig, in Ingres 6.4.

Ingres 6.4 Parameter	Ingres Parameter	Type
Log buffers in memory	buffer_count	Log

Ingres 6.4 Parameter	Ingres Parameter	Type
Transactions in the logging system	tx_limit	Log, Derived
Databases in the logging system	database_limit	Log, Derived
Maximum C.P. interval for invoking the archiver	archiver_interval	Log, Derived
Block size of the log file	block_size	Log
Log-full limit	full_limit	Log
Percentage of log for consistency point	cp_interval	Log, Derived
Force-abort limit	force_abort_limit	Log, Derived
Size of the lock hash table	hash_size	Lock, Derived
Size of the resource hash table	resource_hash	Lock, Derived
Maximum number of locks in the locking system	lock_limit	Lock, Derived
Maximum number of lock lists	list_limit	Lock, Derived
Maximum number of locks per transaction	per_tx_limit	Lock

Notes on Specific Parameters

Note the following:

- There is no 6.4 equivalent to the Ingres log_writer parameter. Although the log_writer default is 1, it is usually advantageous to start your Ingres installation with log_writer set to 4 or 5. If you are using dual logging, double the setting.
- The default rule for computing lock_limit (and the new parameter resource_limit) tend to compute very high numbers—hundreds of thousands, or more. You can allow more locks than you did in 6.4. As an initial setting, a doubling of lock_limit is usually more than sufficient.
- Ingres should usually start with a higher log buffer_count setting than did 6.4; if the 6.4 setting was less than 20, start with 20 buffers.
- Configurator may compute a different archiver_interval than you used in 6.4. Carry over the 6.4 setting.

Troubleshooting Upgradedb Problems

This appendix describes how to troubleshoot problems you may encounter when using the upgradedb utility.

Troubleshooting Tips

The best way to avoid problems with the upgradedb utility is to upgrade to the most recent service pack of Ingres, and to follow the upgrade steps carefully.

Note: If you are upgrading to Ingres II versions 2.0 or 2.5, make sure you install the latest patch available for your platform before performing the upgradedb step.

Here are problems that have occasionally occurred when upgrading:

- The upgradedb utility starts to process, and then hangs with no error indication.

This is probably caused by the Remote Command Server interfering with the upgradedb process. Use the `rmcmdstp` command to stop the Remote Command Server.

- The following message occurs: "Product *name* has been made uninstallable by an incompatible dictionary upgrade."

This message is caused by extra or incorrect rows in the front-end catalog `ii_client_dep_mod`. The rows may have been created by very old versions of Ingres. You can ignore this message.

- The following message occurs shortly after the upgradedb utility starts processing a database: "E_SC0206 An internal error prevents further processing of this query."

This message is seen when **upgradedb -all** is used, and the database data ROOT location is not the same as others processed in the same upgradedb run. The `errlog.log` shows the message "E_DM9004_BAD_FILE_OPEN" referencing a filename: `aaaaaaaa.cnf`, shortly before the E_SC0206 message.

This message has occasionally been seen in various versions of `upgradedb`. Simply rerun `upgradedb` for the one failed database, and continue the upgrade.

If something else goes wrong with the `upgradedb` utility, contact Computer Associates Technical Support for help. For problems with a single database, technical support can assist you in restoring the database data files from your system backup and resetting the database information in `iidbdb` so that you can retry `upgradedb`. In the worst case, it may be necessary to restore the entire installation from your system backup, fix the database problem, and redo the upgrade.

Keywords

This appendix lists Ingres keywords and the contexts in which they are reserved. You can use the list to avoid assigning object names that conflict with reserved words.

Note: The keywords in the list do not necessarily correspond to supported Ingres features. Some words are reserved for future or internal use, and to provide backward compatibility.

Table Key

In the tables in this appendix, the column headings have the following meanings:

Non 6.4 – Keywords not included in Ingres 6.4 keyword reserved lists

ISQL (Interactive SQL) – Keywords reserved by the DBMS

ESQL (Embedded SQL) – Keywords reserved by the SQL preprocessors

IQUEL (Interactive QUEL) – Keywords reserved by the DBMS

EQUEL (Embedded QUEL) – Keywords reserved by the QUEL preprocessors

4GL – Keywords reserved in the context of SQL or QUEL in Ingres 4GL routines

Reserved Single Keywords

The following single keywords are reserved.

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

Keyword	Non 6.4	SQL			QUEL		
		ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
abort		*	*	*	*	*	*
activate			*			*	
add		*	*	*			
addform			*			*	
after	*			*			*
all		*	*		*	*	
alter		*		*			
and		*	*		*	*	
any		*	*	*	*	*	
append					*	*	*
array	*			*			
as		*	*		*	*	*
asc		*		*			
at		*	*	*	*	*	*
authorization		*	*				
avg		*	*	*	*	*	
avgu			*		*	*	
before				*			*
begin		*	*	*	*		*
between		*	*	*			
breakdisplay			*			*	
by		*	*	*	*	*	*
byref	*	*		*			*
call			*	*		*	*

Keyword	Non 6.4	SQL			QUEL		
		ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
callframe	*			*			*
callproc	*	*		*			*
cascade	*	*	*				
case	*	*	*	*			
cast	*	*					
check		*	*	*			
clear			*	*		*	*
clearrow			*	*		*	*
close		*	*		*		
coalesce	*	*					
column		*	*	*		*	
command			*			*	
comment	*			*			
commit		*	*	*			
committed	*	*	*	*			
connect			*				
constraint	*	*	*	*			
continue		*	*				
copy		*	*	*	*	*	*
copy_from	*	*					
copy_into	*	*					
count		*	*	*	*	*	
countu			*		*	*	
create		*	*	*	*	*	*
current		*	*				
current_user	*	*	*				
currval	*	*	*	*			
cursor		*	*				
cycle	*	*	*	*			

Keyword	Non 6.4	SQL			QUEL		
		ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
datahandler	*		*				
dbms_password	*		*	*			
declare		*	*	*			*
default	*	*	*	*			*
define	*	*			*		*
delete	*	*	*	*	*	*	*
deleterow			*	*		*	*
desc				*			
describe	*	*	*				
descriptor			*				
destroy					*	*	*
direct	*			*			*
disconnect			*				
display			*	*		*	*
distinct		*	*	*			
distribute	*				*		
do		*		*			*
down			*			*	
drop		*	*	*			
else		*		*			*
elseif		*		*			*
enable	*			*			
end		*	*	*	*	*	*
end-exec	*		*				
enddata			*			*	
enddisplay			*			*	
endfor		*	*	*			
endforms			*			*	
endif		*		*			*

Keyword	Non 6.4	SQL			QUEL		
		ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
endloop		*	*	*		*	*
endrepeat		*	*	*			
endretrieve						*	
endselect			*				
endwhile		*		*			*
escape		*	*				
except	*	*					
exclude	*				*		
excluding	*	*	*		*		
execute		*	*	*	*		
exists		*	*	*			
exit				*		*	*
fetch		*	*				
field			*			*	
finalize			*			*	
first		*	*	*			
for		*	*	*	*	*	
foreign	*	*	*				
formdata			*			*	
forminit			*			*	
forms			*			*	
from		*	*	*	*	*	*
full	*	*	*	*			
get	*			*			
getform			*			*	
getoper			*			*	
getrow			*			*	
global	*	*	*	*			
goto			*				

Keyword	Non 6.4	SQL			QUEL		
		ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
grant		*	*	*			
granted	*		*	*			
group		*	*	*			
having		*	*	*			
help			*		*	*	
help_forms	*			*			*
help_frs			*			*	
helpfile			*	*		*	*
identified			*	*			
if		*		*			*
iimessage	*		*			*	
iiprintf	*		*			*	
iiprompt	*		*			*	
iistatement	*					*	
immediate		*	*	*			*
import	*	*					
in		*	*	*	*	*	
include			*		*		
increment	*	*	*	*			
index		*	*	*	*	*	*
indicator			*				
ingres						*	
initial_user	*	*	*				
initialize			*	*		*	*
inittable			*	*		*	*
inquire_equel						*	
inquire_forms	*			*			*
inquire_frs			*			*	
inquire_ingres	*		*	*		*	*

Keyword	Non 6.4	SQL			QUEL		
		ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
inquire_sql			*	*			
insert		*	*	*			
insertrow			*	*		*	*
integrity		*	*		*		*
intersect	*	*					
into		*	*	*	*	*	*
is		*	*	*	*	*	*
isolation	*	*	*	*			
join	*	*	*				
key	*		*	*			*
leave		*	*	*			
left	*		*	*			
level	*	*	*		*	*	
like		*	*				
loadtable			*	*		*	*
local	*	*					
max		*	*	*	*	*	
maxvalue	*	*	*	*			
menuitem			*			*	
message		*	*	*		*	*
min		*	*	*	*	*	
minvalue	*	*	*	*			
mode	*			*			*
modify		*	*	*	*	*	*
module	*	*					
move	*				*		
natural	*	*	*				
next		*	*			*	
nextval	*	*	*	*			

Keyword	Non 6.4	SQL			QUEL		
		ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
nocache	*	*	*	*			
nocycle	*	*	*	*			
noecho	*			*			*
nomaxvalue	*	*	*	*			
nominvalue	*	*	*	*			
noorder	*	*	*	*			
not		*	*		*	*	
notrim			*			*	
null		*	*	*		*	*
nullif	*	*					
of		*	*	*	*	*	*
on		*	*		*	*	*
only	*	*	*	*	*		*
open		*	*		*		
option		*					
or		*	*		*	*	
order		*	*	*	*	*	*
out			*			*	
outer	*	*	*	*			
param						*	
partition	*		*				
permit		*	*		*		*
prepare		*	*				
preserve	*	*	*				
primary	*		*	*			
print			*		*	*	
printscreen			*	*		*	*
privileges		*					
procedure		*	*	*			*

Keyword	Non 6.4	SQL			QUEL		
		ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
prompt			*	*		*	*
public		*	*				
purgetable	*		*	*		*	*
putform			*			*	
putoper			*			*	
putrow			*			*	
qualification	*			*			*
raise	*	*		*			
range					*	*	*
rawpct	*	*	*	*			
read	*	*	*		*		
redisplay			*	*		*	*
references	*	*	*	*			
referencing		*		*			
register		*	*	*	*	*	*
relocate		*	*	*	*	*	*
remove		*	*	*		*	*
rename	*				*		
repeat		*	*	*		*	*
repeatable	*	*	*				
repeated			*	*			
replace					*	*	*
replicate	*				*		
restart	*	*	*	*			
restrict	*	*	*				
result	*		*				
resume			*	*		*	*
retrieve					*	*	*
return		*		*			*

Keyword	Non 6.4	SQL			QUEL		
		ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
revoke		*	*	*			
right	*		*	*			
role	*		*	*			
rollback		*	*	*			
row		*	*	*			
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					*	*	*

Keyword	Non 6.4	SQL			QUEL		
		ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
sql		*					
start	*	*	*	*			
stop			*				
submenu			*			*	
substring		*	*				
sum		*	*	*	*	*	
sumu			*		*	*	
system	*			*			*
system_ maintained	*	*	*		*	*	
system_user	*	*	*				
table		*	*				
tabledata			*			*	
temporary	*	*	*				
then		*	*	*			*
to		*	*		*	*	*
type	*			*			
uncommitted	*	*	*	*			
union		*	*	*			
unique		*	*	*	*	*	*
unloadtable			*	*		*	*
until		*	*	*	*	*	*
up			*			*	
update		*	*	*	*		
user		*	*	*			
using		*	*				
validate			*	*		*	*
validrow			*	*		*	*
values		*	*	*			

Keyword	Non 6.4	SQL			QUEL		
		ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
view		*	*		*		*
when	*	*	*				
whenever			*				
where		*	*	*	*	*	*
while		*					*
with		*	*	*	*	*	*
work		*		*			
write	*	*	*	*			

Reserved Double Keywords

The following double keywords are reserved.

Keyword	Non 6.4	SQL			QUEL		
		ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
add privileges	*			*			
after field	*			*			*
alter default	*		*	*			
alter group		*	*	*			
alter location	*	*	*	*			
alter profile	*	*					
alter role		*	*	*			
alter security_audit	*	*	*	*			
alter_sequence	*	*	*	*			
alter table	*		*	*			
alter user	*	*	*	*			
array of	*			*			
base table structure		*					
before field	*			*			*

Keyword	Non 6.4	SQL			QUEL		
		ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
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		*		*			
create integrity	*	*		*			
create link		*	*				
create location	*	*	*	*			
create permit	*	*		*			
create procedure	*			*			
create profile	*	*	*	*			
create role		*	*	*			
create rule		*	*	*			
create security_alarm	*	*	*	*			
create sequence	*	*	*	*			
create synonym	*	*	*	*			
create user	*	*	*	*			

Keyword	Non 6.4	SQL			QUEL		
		ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
create view	*	*		*			
cross join	*	*	*	*			
curr value	*	*					
current installation	*			*			
current value	*	*	*	*			
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	*						*
direct connect			*	*		*	*
direct disconnect			*	*		*	*
direct execute			*	*			*
disable security_audit	*	*	*	*			
disconnect current	*			*			
display submenu	*			*			*
drop dbevent		*	*	*			
drop domain	*		*				

Keyword	Non 6.4	SQL			QUEL		
		ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
drop group		*		*			
drop integrity	*	*		*			
drop link		*	*	*			
drop location	*	*	*	*			
drop permit	*	*		*			
drop privileges	*			*			
drop procedure	*			*			
drop profile	*	*	*	*			
drop role		*	*	*			
drop rule		*	*	*			
drop security_alarm	*	*	*	*			
drop sequence	*	*	*	*			
drop synonym	*	*	*	*			
drop user	*	*	*	*			
drop view	*	*		*			
each row	*		*				
each statement	*		*				
enable security_audit	*	*	*	*			
end exclude	*		*				
end transaction		*	*	*	*	*	*
exec sql	*		*				
execute immediate	*			*			
execute on	*			*			
execute procedure	*			*			
foreign key	*	*		*			
for deferred		*			*		
for direct		*			*		
for readonly		*			*		
for retrieve	*				*		

Keyword	Non 6.4	SQL			QUEL		
		ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
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	*	*		*			
left outer	*	*		*			
modify table	*			*			
next value	*	*	*	*			
no cache	*	*	*	*			
no cycle	*	*	*	*			
no maxvalue	*	*	*	*			
no minvalue	*	*	*	*			

Keyword	Non 6.4	SQL			QUEL		
		ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
no order	*	*		*			
not like	*	*		*			*
not null	*				*		
on commit	*	*	*	*			
on current	*	*					
on database		*		*			
on dbevent		*		*			*
on location	*	*		*			
on procedure	*	*					
on sequence	*	*					
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	*			*			*

Keyword	Non 6.4	SQL			QUEL		
		ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
replace cursor			*		*	*	*
result row	*	*	*	*			
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 hash	*	*			*		
set io_trace	*	*			*		

Keyword	Non 6.4	SQL			QUEL		
		ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
set j_freesz1	*	*			*		
set j_freesz2	*	*			*		
set j_freesz3	*	*			*		
set j_freesz4	*	*			*		
set j_sortbufsz	*	*			*		
set jcpufactor	*				*		
set joinop	*	*			*		
set journaling	*	*			*		
set lock_trace	*	*			*		
set lockmode	*	*			*		
set log_trace	*	*			*		
set logdbevents	*	*					
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 nohash	*	*					
set noio_trace	*	*			*		
set nojoinop	*	*			*		
set nojournaling	*	*			*		

Keyword	Non 6.4	SQL			QUEL		
		ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
set nolog_trace	*	*			*		
set nolog_trace	*	*			*		
set nologdbevents	*	*					
set nologging	*	*			*		
set nomaxconnect	*	*			*		
set nomaxcost	*	*			*		
set nomaxcpu	*	*			*		
set nomaxidle	*	*			*		
set nomaxio	*	*			*		
set nomaxpage	*	*			*		
set nomaxquery	*	*			*		
set nomaxrow	*	*			*		
set noofflatten	*	*					
set nooptimizeonly	*	*			*		
set noparallel	*	*					
set noprintdbevents	*	*					
set noprintqry	*	*			*		
set noprintrules	*	*					
set noqep	*	*			*		
set norowlabel_visible	*	*					
set norules	*	*					
set nosql	*				*		
set nostatistics	*	*			*		
set notrace	*	*			*		
set nunicode_substitution	*	*					
set ojflatten	*	*					
set optimizeonly	*	*			*		
set parallel	*	*					

Keyword	Non 6.4	SQL			QUEL		
		ISQL	ESQL	4GL	IQUEL	EQUEL	4GL
set printdbevents	*	*					
set printqry	*	*			*		
set printrules	*	*					
set qbufsize	*	*			*		
set qep	*	*			*		
set query_size	*	*			*		
set random_seed	*	*			*		
set result_structure	*	*			*		
set ret_into	*	*			*		
set role	*	*					
set rowlabel_visible	*	*					
set rules		*					
set session	*	*			*		
set sortbufsize	*	*			*		
set sql	*				*		
set statistics	*	*			*		
set trace	*	*			*		
set transaction	*	*					
set unicode_substitution	*	*					
set update_rowcount	*	*			*		
set work	*	*					
system user	*			*			
to group		*		*			
to role		*		*			
to user		*	*	*			
user authorization	*			*			
with null					*		
with short_remark	*	*					

Other Reserved Keywords

The following reserved keywords are only in the context of a WITH PARTITION= clause.

automatic	partition
hash	range
list	to
null	values
on	with

Features Introduced in Advantage Ingres 2.6

Advantage Ingres 2.6 has several new enhancements including the following:

- User-visible language enhancements
- Increased maximum size of character data types
- User-visible DBA enhancements
- Internal performance enhancements
- Locking system performance improvements
- Logging system performance improvements
- Buffer manager performance improvements
- Operating system integration
- Ingres ICE enhancements
- ODBC enhancements
- JDBC enhancements
- Support for Unicode
- New character sets to support Euro currency symbol

User-Visible Language Enhancements

Enhancements have been made to the internal performance that concern row producing procedures.

Row Producing Procedures

This enhancement to the Advantage Ingres database procedure language addresses the ability of Advantage Ingres to read and return to the caller multiple rows from a select statement.

With server-executed database procedures, the program logic of the procedure is executed entirely in the server address space. Multiple SQL DML requests are executed in a single invocation of the procedure, with only one interaction with the client application. The ability to process and return multiple “rows” of some composite data types with a single call to a server-resident database procedure adds to the potential for improved performance of an application.

In a typical computing environment with applications executing on a variety of computers throughout a network, this approach can significantly reduce the footprint of the client application and the traffic across the network.

SUBSTRING Function

The ANSI compliant SUBSTRING function has been added to the SQL syntax. The SUBSTRING function is often easier to use than combinations of LEFT and SHIFT functions that Ingres traditionally supported. The syntax is:

substring(*string-expr* **from** *start-column* **for** *length*)

and the **for** *length* clause is optional.

New Aggregate Functions

Additional aggregate functions for statistical analysis have been added:

- The function **corr** computes a correlation coefficient
- Functions **covar_samp** and **covar_pop** compute covariance
- A collection of **regr_xx** functions generate regression analysis results

For details, see the *SQL Reference Guide*.

Increased Maximum Size of Character Data Types

Prior to Advantage Ingres 2.6, character data types were limited to a maximum size of 2000 bytes; this restriction was imposed when the maximum size of a row was limited to 2 KB. This limit is increased to 32000 bytes, the maximum row size supported in Advantage Ingres 2.6.

User-Visible DBA Enhancements

Enhancements have been made to internal performance that concern auditdb utility, copydb utility, raw location support, and GatherWrite threads.

Usermod Utility

Advantage Ingres now includes a usermod utility that allows users to run the modify commands on user tables. Like sysmod, which modifies system catalogs, this utility is useful for maintaining user tables on a regular basis.

Running this utility regularly, or when the table has excess overflow pages, improves performance of user applications.

Auditdb Utility

Various enhancements to the auditdb utility required by Journal Analyzer are included:

- Specification of fully qualified table names.
- Correct formatting of the output for -aborted_transaction when used with -b and -e flags.
- Corrected -aborted_transaction flag, allowing auditdb to write correct format for BT and ET records.
- Savepoint information in the auditdb output. This is achieved by printing out the abortsave record, which contains the LSN of the aborted savepoint.
- The order of output for lsn low/high fields for the ASCII output of auditdb, allowing the high lsn to be printed before low LSN.
- Two new auditdb options: -start_lsn=<LSN> -end_lsn=<LSN> for non -all cases.

For the syntax of the auditdb command, see the *Command Reference Guide*.

Copydb Utility

The copydb utility is modified to include several options and flags that modify the copy.in and copy.out scripts based on user requirements. The user can specify the order in which the copy and modify statements are written to the copy.in script, for example, whether to copy the data into the tables and then run a modify statement or the other way around. Other examples include the ability to remove hard-coded paths to the copy scripts, exclusion of location names, and exclusion of user-specific permissions such as grant statements.

Raw Location Support

Advantage Ingres 2.6 adds initial support for raw data locations on UNIX platforms. Raw data locations provide dramatic performance improvements over cooked locations. In this release, only one table may occupy any given raw location. Many raw locations can exist on a single raw disk slice.

GatherWrite Threads

A new internal thread type, GatherWrite, is used by the Advantage Ingres buffer manager during operations that require the flushing of multiple buffers from the cache such as write behind, consistency points, and table purges. This feature is only available on platforms that offer `writenv()` support. Consult the appropriate Readme file to determine whether this feature is supported on your platform.

This feature is enabled on a per-server basis using the `gather_write` parameter in Configuration-By-Forms. The default setting is ON.

XML Import/Export Utility

XML is as a cross-platform, software- and hardware-independent tool for transmitting information. The XML import/export utility imports and exports XML data from Ingres tables to and from XML files. For the syntax of the XML import/export utility, `xmlimport`, see the *Command Reference Guide*.

Journal Analyzer

The Journal Analyzer is a powerful graphical tool that provides an interface to the journal files. You can use the Journal Analyzer to recover data from the journals and to apply journaled transactions to other databases, both local and remote. For information on the Journal Analyzer utility, see the *System Administrator Guide*.

Import Assistant

The Import Assistant is a wizard that simplifies the task of importing data from a standard file format to an Advantage Ingres database. For information on the Import Assistant utility, see the online help.

Automated Creation of Location Directories

Before Version 2.6, the Ingres DBA had to manually create the directories for alternate locations as prescribed in the *Database Administrator Guide*. This step had to be performed prior to creating a Location with ACCESSDB, or could be deferred if Locations were created using EXEC SQL CREATE LOCATION syntax. To circumvent directory permissions problems, ACCESSDB had to be run by the Ingres user whenever Locations were created, altered, or extended.

This process is clarified and simplified in Advantage Ingres 2.6 with the following changes:

- The Ingres server performs all manipulations of Location directories. This resolves the permissions problems of earlier releases and allows any ACCESSDB user with the “maintain_locations” privilege to create, alter, or extend Locations.
- The server automatically creates Location directories when a CREATE/ALTER LOCATION statement is executed, whether by ACCESSDB or user-invoked SQL. Because only missing directories are created, the DBA retains the ability to manually create as much, or all, of the Location path as wanted before creating the Location.

Using the example from the section Creating an Area in UNIX in the *Database Administrator Guide*, the following directories will be verified or created automatically during the execution of:

```
CREATE LOCATION new_loc WITH AREA='/otherplace/new_area', USAGE=(DATABASE)
```

Perms	Directory
	/otherplace
755	/otherplace/new_area
755	/otherplace/new_area/ingres
700	/otherplace/new_area/ingres/data
777	/otherplace/new_area/ingres/data/default

Note the following:

- Permissions are **not** changed for extant directories.
- The top-level directory “/otherplace” must exist and will **not** be created by the server.
- Raw location directories (UNIX only) cannot be automatically created and must be made with the MKRAWAREA utility, which must be run by “root.” The Locations may be created prior to MKRAWAREA but a warning will be issued noting that the utility must be run prior to their use.

Remote Command Server Enhancements

Users commonly encounter problems running utilities that require exclusive access to the iidbdb database because the Remote Command Server process (rmcmd) keeps a session open on this database. To counter this problem, rmcmd now attaches to imadb instead of iidbdb; imadb is a system database that contains no historical data; it is rarely backed up and requires little or no maintenance.

Microsoft Transaction Server Support

Support for tightly coupled XA threads and shared lock lists is now available to support Microsoft Transaction Server, using the ODBC 3.5 driver.

Concurrent Rollback

The concurrent recovery of multiple transactions is now possible.

Internal Performance Enhancements

Enhancements have been made to the internal performance that concern aggregate sort nodes, composite histograms, and optimizer support for hash joins.

Aggregate Sort Nodes

Improvements to aggregate handling allows Advantage Ingres to better support data-mining products such as CleverPath OLAP, which make extensive use of data aggregation.

Previous versions required a sort before doing grouping and aggregation. Advantage Ingres 2.6 now does grouping with hash bucketing instead of sorting. Hash grouping is usually faster than sorting. Other internal refinements streamline the calculation of common aggregates, reducing the amount of CPU time needed.

Composite Histograms

The composite histograms enhancement allows the creation of composite or multi-column histograms that model much more accurately the dependence of the values of one column on another, and lead to far better selectivity estimates and, ultimately, to better query plans.

Optimizer Support for Hash Joins

Hash joins have been implemented in Advantage Ingres 2.6. A hash join is one in which a hash table is built with the rows of one of the join sources by hashing on the key columns of the join. The rows of the other join source are then read and hashed into the table on their key columns. The hashing of the second set of rows quickly identifies pairs of joining rows. This technique requires no index structures on the join columns (as does KEY join), nor does it require sorting on the join columns (as does merge join).

Locking System Performance Improvements

A number of improvements have been made to the locking system to eliminate or minimize bottlenecks identified when running various performance tests.

Preallocated RSB/LKBs

Each resource RSB now has an embedded lock block (LKB), removing the need for a separate, contentious LKB allocation every time a new resource is allocated.

An LLB stash of LKBs is also maintained, similar to the RSB stash.

When an RSB or LKB is freed, it is returned to the LLB's stash; when the lock list itself is freed, all stashed RSB/LKBs are returned to the free pool.

Miscellaneous Locking System Improvements

The following miscellaneous locking system improvements are included in Advantage Ingres 2.6:

- The number of RSB waiters and converters are now maintained in the RSB.
- The deadlock wait-for graph lock (lkd_dlock_lock) does not need to be held if the RSB has neither waiters nor converters.

- The LKREQ built in the stack does not need to be copied to the LKB indiscriminately.
- When a lock request is blocked, the blocker's identity is now saved in the LKREQ and formatted in SYS_ERR only when the request fails.

Logging System Performance Improvements

A number of improvements to the logging system eliminate or at least minimize bottlenecks identified when running various performance tests. These changes include elimination of contentious `current_llb_mutex`, faster log forces through forcing only what needs forcing, and improved concurrency potential (fast resume).

Buffer Manager Performance Improvements

A number of improvements have been made to the buffer manager to eliminate or minimize bottlenecks identified when running various performance tests. These include:

- Removal of stats for fixed priority pages. In Advantage Ingres 2.6, stats are tracked by buffer page type for better analysis of the BM's LRU algorithm.
- Raising a buffer's priority each time it is fixed; previously it was raised only when newly fixed.

Operating System Integration

Enhancements have been made to the internal performance that concern 64-bit operating systems and operating system thread implementation on Linux.

64-Bit Operating Systems

Now that Microsoft, Sun, HP, and Linux vendors have produced 64-bit versions of their operating systems, we are providing a 64-bit build of Advantage Ingres on these platforms. Every effort is made to exploit large memory and files in these 64-bit environments.

Operating System Thread Implementation on Linux

Advantage Ingres 2.6 provides support for operating system threads in Linux environments including Intel, Alpha, S/390, and IA64. Operating system threads perform better in most circumstances than the internal Advantage Ingres threading model.

Ingres ICE Enhancements

Ingres/ICE development environment and setup and configuration are addressed in Advantage Ingres 2.6 through integration with an existing Web application development environment.

Development Environment

The ICE macro DTD can be used with an XML-aware editor to provide a development environment for Ingres/ICE application development. A converter has been added to take new macro syntax into the old macro syntax during page registration.

ODBC Enhancements

The ODBC driver has been updated to Version 3.5.

Supported Functions

The Advantage Ingres ODBC driver supports all level one functions, as well as the following level two functions:

- SQLExtendedFetch (through Microsoft Cursor Library only)
- SQLForeignKeys
- SQLMoreResults
- SQLNumParam
- SQLPrimaryKeys
- SQLProcedureColumns
- SQLProcedures
- SQLSetPos (through Microsoft Cursor Library only)

Unavailable Features

The Advantage Ingres ODBC driver does not provide the following features as of Release 2.6:

- Executing functions asynchronously
- Translation DLL
- Support for Advantage Ingres SQL COPY TABLE command
- Support for Advantage Ingres SQL SAVEPOINT command

JDBC Enhancements

The following JDBC 2.0 extensions have been added to Advantage Ingres:

- Compatibility with GA release (protocol levels)
- Execution in JDK 1.1 environment due to a new driver
- Batch processing
- javax—two-phase commit
- javax—client connection pooling
- Updateable result sets

The following are new features:

- Support for Advantage Ingres intervals
- Coalescing statement IDs
- Utilization of VNODE passwords
- Local connections without passwords
- Support for procedure table parameters
- Support for row producing procedure

Support for Unicode

This release contains the first phase of Advantage Ingres support for Unicode; further Unicode support will be added in future releases.

In this release, the DBMS supports three new data types:

- `nchar`
- `nvarchar`
- `long nvarchar`

These data types store character data using two bytes for each character. Collation of these data types uses the standard collation algorithm as defined by the Unicode organization, and the data types can be used in indexes and database statistics.

The native two-byte (UCS2) format is supported and maintained through the entire process, from the front-end application, through the DBMS, to the data representation on disk.

The embedded SQL preprocessor for C and C++ supports declaration of `wchar_t` variables, which are assumed to contain multi-byte Unicode character strings.

OpenAPI version 3 was added to indicate support for these new data types.

VDBA also supports the new data types.

Support for the ODBC and JDBC drivers is present through their normal Unicode interfaces.

These new data types are not supported in any of the character-based tools or any of the terminal monitors.

This release does not support coercion between Unicode data types and non-Unicode data types such as `char` and `varchar`.

New Character Sets to Support Euro Currency Symbol

Two new character sets that contain the Euro currency sign (€, Unicode U+20AC) are added: IS885915 and WIN1252.

To set the money format to the Euro currency symbol you must issue the following command:

```
ingsetenv II_MONEY_FORMAT L:€
```

Alternatively, you can set this value in the Advantage Ingres Visual Manager (IVM).

Windows

WIN1252 corresponds to Windows code page 1252 Latin 1. This is the common character set of most American and Western European Windows PCs, and includes the Euro sign. Users wishing to use the Euro symbol in a Windows GUI environment need to select the WIN1252 character set at installation time. To set this code page in a Windows command prompt environment, you must issue the following Windows command:

```
chcp 1252
```

The default font in a Windows command prompt does not provide support for the Euro currency symbol. For a workaround, set the font to Lucida Console. The Lucida Console font has moved the line drawing characters, used in Advantage Ingres forms, into an area not accessible to Advantage Ingres binaries, so we have provided rudimentary line drawing in the IBMPCD terminal entry. To set this terminal type, you must either issue the following command:

```
ingsetenv TERM_INGRES IBMPCD
```

or set TERM_INGRES through IVM or specify this terminal type at install time.

**UNIX**

IS885915 corresponds to the ISO 8859-15 Latin 9-character set that is almost identical to the ISO 8859-1 Latin 1 set, except for eight characters; chief among them is the Euro currency sign (€, Unicode U+20AC).

If you have an existing installation and would like to change the character set, be aware that this is not typically supported because the new character set could display existing characters in your databases incorrectly. However, since the ISO 8859-15 only has eight characters that are different from ISO 8859-1, if you can verify that none of the eight characters are already present in your databases, you could safely change the set (by changing II_CHARSETxx).

The following table details these differences and provides the corresponding Unicode character names:

Hex		ISO 8859-1		ISO 8859-15
A4	■	Currency symbol	€	Euro sign
A6		Broken bar	Š	Latin capital letter with caron
A8	¨	Diaeresis	š	Latin small letter with caron
B4	'	Acute accent	Ž	Latin capital letter Z with caron
B8	,	Cedilla	ž	Latin small letter Z with caron
BC	¼	Vulgar fraction one quarter	Œ	Latin capital ligature OE

Hex		ISO 8859-1		ISO 8859-15
BD	½	Vulgar fraction one half	Œ	Latin small ligature oe
BE	¾	Vulgar fraction three quarters	Ÿ	Latin capital letter Y with diaeresis

Differences Between ISO 8859-1 and ISO 8859-15 Character Sets ■

Features Introduced in Ingres II 2.5

This appendix describes the features and enhancements introduced in Ingres II 2.5, including:

- Sort enhancements
- ANSI/ISO constraint enhancements
- Large cache support
- Dynamic Write Behind threads
- Partitioned transaction log file
- Optimizedb enhancements
- Read-only database support
- New SQL functionality
- Extended date support
- Large file support
- Large catalogs
- Row locking for system catalogs
- Update mode locking
- Query optimization enhancements
- Ingres Star features
- Ingres Net features
- Ingres ICE features
- Visual DBA features
- Replicator enhancements
- OpenAPI enhancements

Sort Enhancements

Changes were made to improve the performance of both the in-memory (QEF) sort and disk (DMF) sort of Ingres II.

QEF Sort Enhancements

QEF was improved by fine-tuning the sort algorithm, resulting in fewer comparisons between sort rows. The sort algorithm is a major consumer of CPU time in a sort. QEF was also improved with a change that results in the rows being partially sorted as they arrive in the sort.

This change introduces two distinct benefits:

- Duplicate rows are detected and discarded more quickly from duplicate removal sorts (as required by “select distinct...”). This in turn increases the number of rows that can be processed in memory for a duplicates removal sort, avoiding more expensive disk sorts in many instances.
- The first rows in the sort sequence can be returned before the remaining rows are completely sorted. Tests show that the first sorted row is available with as few as 20% of the overall comparisons required to complete the sort. This means that browsing or scrolling applications see the first set of rows in less time than before.

DMF Sort Enhancements

The first set of DMF sort enhancements also involve fine-tuning of the sort algorithms, which should result in a 5 to 10 percent reduction in CPU time of typical sorts. As with the QEF sort, duplicate rows are detected and discarded sooner in duplicates removal sorts. This should result in smaller disk work files and faster overall sort performance.

Prior to Release 2.5, the entire result of a DMF sort was spooled to an internal temporary table before the sorted rows were returned to the caller. In Ingres II 2.5, the temporary table has been eliminated and the rows are returned directly from the sort structures to the caller. This has the same effect as the early return of sorted rows described above for the QEF sort. That is, the first rows should be returned much sooner than they were in previous releases.

The final DMF sort enhancement is the introduction of a “parallel sort” technique. Sorts that exceed a user-configurable threshold spawn additional threads. The sort is split up and its rows delivered to the sub-threads for sorting. The sorted subsets of the rows are then delivered back to the parent thread executing the query, where they are merged to form a single sorted stream of rows.

On multi-CPU machines, this results in a significant reduction in the elapsed time required to sort (between 25 to 50 percent in testing). Even single CPU machines benefit somewhat, because sort I/O and sort computation can be overlapped. An added benefit to the parallel sort technique is that it is encapsulated within the DMF sort. This sort is used for the execution of queries with sorting requirements (such as for order by, group by, and distinct requests, or for implementing certain join algorithms). However, it is also used to sort rows for index creation or update in modify, create index, and copy operations. All users of the DMF sort derive the performance benefit of the parallel sort.

Parallel Sort Techniques

The “parallel sort” technique outlined above is used to sort rows for parallel index creation, greatly reducing the time taken for index creation in multi-CPU environments.

ANSI/ISO Constraint Enhancements

Ingres referential and unique/primary key constraints result in the creation of indexes “under-the-covers” to improve the performance of the constraint enforcement mechanisms. Prior to Release 2.5, these indexes were plain B-tree indexes stored in the default location of the database. However, B-tree is not always the best choice (for example, hash is better for many unique key applications), and use of the default location can degrade performance if many large indexes are created.

Ingres II 2.5 solves these and other problems by including a “with” clause for constraint definition. The “with” clause allows the overriding of default index options with anything normally coded in an index creation “with” clause. For example, the index structure and location, as well as fillfactor and other index options can be explicitly specified for each constraint. The “with” clause applies to column and table constraints defined with both the create and alter table statements. A unique/primary key constraint can be generated to use the base table structure for its enforcement rather than a separate secondary index.

Ingres II 2.5 also introduces the ANSI/ISO notion of referential actions for the definition of referential (foreign key) constraints. In releases prior to Ingres II 2.5, the attempt to delete a referenced row for which matching referencing rows exist, or to update the primary key of a referenced row to some other value while matching referencing rows still exist for the old value, was met with an error and the request was aborted. Either operation had to be preceded by a delete of the matching referencing rows or an update of the foreign keys to some value that exists in another referenced row.

Ingres II 2.5 allows the definition of referential actions for each referential constraint, which defines alternative actions to be taken in the circumstances defined above. A separate action can be defined for both the delete case (deletion of a referenced row with matching referencing rows) and the update case (updating the key of a referenced row with matching referencing rows). The options include *cascade*, in which case the delete or update is cascaded to the matching referencing rows (so that the referencing rows are also deleted or updated to the same value), and *set null*, in which case the foreign key of the matching referencing rows is set to null. These actions permit a more complete definition of the semantics of the referential relationship and allow the application to execute more efficiently.

Large Cache Support

In Ingres II 2.5, the total number of pages in all caches has been revised from an un-enforced limit of 65536 to $2^{32}-1$. Ingres II 2.5 supports a 4 GB cache.

In previous releases, when those pages belonging to a specific table needed to be located, the buffer manager sequentially searched every buffer in every cache to find them. Even in installations with small caches, this was an expensive operation, especially in those frequent instances in which there were no table-pages in any cache. This operation occurred, for example, when a table's TCB was about to be released, typically when all referencing transactions had **no immediate need to use the table**. Delays caused by this operation could show up in unexpected situations, such as the use of statement level rules.

In Ingres II 2.5, a cross-cache table hash queue has been added to the buffer manager to which pages are added as they are faulted in and removed when they are tossed. Thus, when the need to know a table's pages arises, a hash on the table's database ID and table ID is made and that list searched for matching pages. This change results in a significant decrease in the number of cache pages visited and is most dramatic in installations configured with very large or multiple caches.

Dynamic Write Behind Threads

In releases prior to Ingres II 2.5, a fixed number of Write Behind threads were configured in each server in an installation and initiated when the server started. These threads served all caches and were awakened when the number of modified pages in any cache exceeded a predefined threshold. In a shared cache environment, all Write Behind threads in all servers were simultaneously activated when this threshold was reached. This led to a “thundering herd” phenomenon in which n Write Behind threads concurrently pounded through the caches, competing for modified pages to flush.

The optimum number of Write Behind threads is the minimum number required to:

- Maintain the modified buffer count below the Write Behind start threshold
- Supply sufficient free pages to avoid synchronous writes.

The optimum number of Write Behind threads varies with the instantaneous demand for free pages in a particular cache; it always begins at one when the threshold is first breached and a Write Behind event signaled. In Ingres II 2.5, if the single Write Behind thread is unable to keep up with the demand, additional Write Behind threads are created until either equilibrium is achieved or the upper limit on thread numbers is reached. If better than equilibrium is achieved (more modified pages are being freed than are in demand), the excess Write Behind threads terminate one-by-one, while the remaining threads continue to monitor the free buffer demand to achieve the write-behind end threshold.

Ingres II 2.5 introduces cache-specific Write Behind threads, resulting in increased concurrency and eliminating the chance of free page starvation.

Partitioned Transaction Log File

The structure of the log file in Ingres II 2.5 is changed from a single file to 1-16 logically striped files of equal size. Properly configured, partitioning in this manner encourages better log performance by spreading disk contention across multiple disks instead of concentrating it on a single device. Ingres system administration is made simpler, as the transaction log file can be expanded by adding another partition, instead of resizing an existing file or partition. The full log file paths are now defined through Configuration Manager or CBF rather than through the symbol table.

Optimizer and Optimizedb Enhancements

A variety of enhancements have been made to optimizedb.

The flag `-zlr` causes optimizedb to retain the original repetition factor when rebuilding an existing histogram. This is useful when a histogram (and its repetition factor) is built once by reading the whole table, then updated later using sampling (which can produce inaccurate repetition factors).

A minor bug was fixed to allow the `"l"` flag to request an exclusive lock on the database during optimizedb processing (just as for other command line utilities).

The current limit of 1000 parameters coded in a separate file using the `-zf` parameter has been lifted. There is now no limit to the number of such parameters.

An `-o` filename option (similar to that in `statdump`) has been added to optimizedb. It creates a `statdump`-style output file, which can then be input back to optimizedb with the `-i` filename option. But more importantly, it does not update the catalog `iistatistics` and `iihistogram` tables. This allows a busy shop to construct histograms at anytime, with no worry about update conflicts. Then at a convenient later time, optimizedb can be run with the `-i` option to add the histograms to the catalog.

In addition to the flag enhancements, an enhancement has been introduced to allow more accurate histograms to be built on columns with significant skew in their value sets. Specifically, a column with many distinct values, which would generate an inexact histogram, now produces exact cells for values that occur significantly more than the average. This permits much more accurate estimates in the compilation of queries with restrictions on such columns, and, therefore, better query plans.

The query compiler has been enhanced to compile more efficient strategies for complex queries involving aggregate views and unions.

Read-only Database Support

Ingres II 2.5 provides the ability to distribute a read-only database on a CD-ROM or other read-only media.

Example

For example, to create a read-only database called `mydatabase` on UNIX:

1. Log in as the installation owner.
2. Change location to the staging directory:

```
cd /stagingarea
```

3. Create directory and subdirectories:

```
mkdir ingres  
mkdir ingres/data  
mkdir ingres/data/default
```
4. Place appropriate permissions:

```
chmod 755 ingres  
chmod 700 ingres/data  
chmod 777 ingres/data/default
```
5. Change location to the database files:

```
cd /install/inst1/ingres/data/default
```
6. Copy the database directory and its subdirectories to the new area:

```
cp -r mydatabase /stagingarea/ingres/data/default/
```
7. Copy the directory structure to the CD-ROM or other device:

```
cp -r ingres/data/default/mydatabase /cdrom
```
8. Create a new database location using the create location statement or using the accessdb utility:

```
create location cdromloc with area=/cdrom, usage=(database);
```
9. Use the createdb command to access the read-only database in the installation:

```
createdb -r cdromloc mydatabase
```

New SQL Functionality

New SQL operations have been added, bringing Ingres SQL closer to the SQL standards. Enhancements have been made to the internal performance that concern bit-wise operator support and miscellaneous functions.

Order By/Group By Expression

The ORDER BY and GROUP BY statements now allow an expression instead of being limited to column names. ORDER BY can also reference a column name or expression that is not part of the select result list.

CASE Expression

An ANSI SQL '92 compliant CASE expression has been added. The CASE expression allows if-then-else testing anywhere that an expression is allowed.

There are two syntax forms. The most general CASE expression is:

```
case when boolean-expression then expression  
      when boolean-expression then expression  
      ...  
      else otherwise-expression  
end
```

Each *boolean-expression* is evaluated in turn, and if TRUE, the corresponding **then** expression is the CASE result. If all the *boolean-expressions* are of the form *expr1* = *expr2*, a shorthand form can be used:

```
case expr1 when expr2 then expression  
      when expr3 then expression  
      ...  
      else otherwise-expression  
end
```

Parallel Index Creation

A new variation of the CREATE INDEX statement allows the user to create multiple secondary indexes with a single pass through the base table. After the required base table columns are extracted, the indexes are created in parallel, each one using an independent worker thread. For additional performance, any necessary sorting is performed using the new parallel sort capability.

The new syntax is:

```
create index (index-spec), (index-spec), ...
```

where an **index-spec** looks similar to the original CREATE INDEX statement:

```
(index-name on base-table (column-list) with with-clause)
```

SELECT Enhancement

The SELECT statement now allows the **first *n*** clause in the result list. This clause limits the result returned to the user to the first N rows.

Bit-wise Operator Support

The following functions have been added to Ingres II 2.5 to provide support for bit-wise operations:

bit_add—Logical “add” of two byte operands

bit_and—Logical “and” of two byte operands

bit_not—Logical “not” of two byte operands

bit_or—Logical “or” of two byte operands

bit_xor—Logical “exclusive or” of two byte operands

For all of these bit functions, all operations proceed right to left. The shorter of two operands is padded with hex zeroes on the left. The result is a byte field equal in size to the longer operand.

Aggregate Functions

New aggregate functions have been added:

- **stddev_samp**
- **stddev_pop**
- **var_samp**
- **var_pop**

The **_pop** forms divide by group size, the **_samp** forms divide by group size minus one.

Miscellaneous Functions

The following miscellaneous functions have been added to Ingres II 2.5:

intextract—Extract the number at the given location.

ii_ipaddr—Convert an IP address to a byte 4.

random, randomf—Generate random integer or float8 values

power, ln—ANSI compliant synonyms for ****** and **log** functions.

Several synonyms have been added to existing Ingres data types (such as character long object and clob for long varchar and binary long object and blob for long byte).

Extended Date Support

Ingres II 2.5 allows users to insert dates in the range 01-Jan-0001 to 31-Dec-9999.

Large File Support

A major enhancement to Ingres II 2.5 on operating systems that support 64-bit file systems is the ability to support file sizes greater than 2 GB. This means that larger table, dump, work, journal, and checkpoint files can be accommodated in a single location. It also removes the 2 GB limit on the size of the transaction log file.

Large Catalogs

In this release, system catalogs can use pages larger than 2 KB. As a result, the user does not have to configure a 2 KB cache size in the DBMS for system catalogs.

Row Locking for System Catalogs

For improved concurrency, the Ingres II 2.5 DBMS automatically uses row locking on system catalogs when catalogs are created using pages larger than 2 KB. This feature is keyed from the system default page size, which is configurable through Configuration-By-Forms or Configuration Manager. Createdb creates a database with system catalogs that have the default page size. Running sysmod on an existing database, however, does not automatically convert the system catalogs to use the default page size. The user must use the “with page_size” keyword to achieve this.

Update Mode Locking

Prior to Ingres II 2.5, when the DBMS fetched a row for a cursor mode update, it would acquire an exclusive lock. In serializable mode, this lock would be held until the end of transaction, even when the row was not updated. In this release, the Ingres DBMS acquires an update mode lock for the row that is a candidate for update. If the row is actually updated, the update mode lock is converted to logical exclusive lock; otherwise, it is converted down to shared lock or released, depending on the current isolation level. Update mode locking is now the default for cursor updates.

Value Locking for Serializable Transaction with Equal Predicate

Prior to Ingres II 2.5, the Ingres DBMS would hold a page lock on the leaf pages on B-tree tables for a serializable update transaction, even when using row locking. This was done to prevent other users from inserting a row in the qualified range to be updated. In Ingres II 2.5, the DBMS uses a value locking protocol for serializable transactions with an equal predicate, thereby allowing better concurrency.

Query Optimization and Execution Enhancements

Ingres II 2.5 incorporates a variety of internal enhancements to the compilation and execution of queries.

Ingres Star Features

In Ingres II 2.5 the Ingres Star Server now contains support for the 1Pcplus commit protocol, which allows a single site that is not capable of supporting two-phase commit protocols to participate in a multi-site update within Star. This change allows a single database that is being accessed through an Ingres Enterprise Access gateway that does not support the SQL prepare statement to participate in a multi-site Star update.

Ingres Net Features

GCF has responsibility for authenticating and validating clients for Ingres servers. Previously, security was built around operating system capabilities. The following improvements have been made to Ingres security for Ingres II 2.5:

- Support for third-party security systems such as Kerberos
- Enhancements for data encryption and direct network server connections
- Improved existing security by addressing known problems
- Backward compatibility for existing applications

Support for third-party security systems requires dynamic configuration capabilities since these systems are not a requirement for installation. In a design emulating the emerging standard GSS-API, the Ingres II 2.5 GCF security architecture is built around independent modules called *mechanisms*. Standard default mechanisms are provided for basic Ingres security and backward compatibility. Third-party security systems are supported through additional mechanisms, which are dynamically loaded as needed.

GCF security mechanisms provide the following capabilities:

- User authentication and validation
- Password validation
- Trusted server authentication and validation
- Distributed (single sign-on) authentication and validation
- Data encryption

Management of GCF security has been enhanced with new configuration parameters viewable through CBF and the Configuration Manager. Ingres II 2.5 also sees the addition of attributes to the Ingres/Net VNODE database, and new IMA objects (many of which can be set at runtime) for enhanced IMA support.

Ingres ICE Features

New features added to the internal performance of Ingres/ICE concern security, session management, storage management, and macro language extensions.

Security

The security model for Ingres/ICE has been enhanced to provide additional levels of control for access and for content. The model also assists in the maintenance of large numbers of Internet users by defining profiles and roles.

Profiles enable Internet users to register themselves by automatically transferring a predefined set of privileges to the user when the first login is attempted.

Password authentication for intranets may be specified as OS for use with domain servers and authentication servers.

Session Management

Ingres/ICE uses cookies to identify individual sessions. The session identifier enables maintenance of session context between pages. Sessions have a configurable inactivity timeout that when reached rolls back any uncommitted transactions.

Storage Management

HTML Templates	Document content is made available through template files. In Ingres II 2.5, access to the template files is abstracted to prevent exposure of queries and schema.
Document Cache Management	All files accessed within Ingres/ICE are subject to cache management, which specifies when the file may be closed or removed.

Macro Language Extensions

Macro language extensions in Ingres II 2.5 include:

- User identification
- State maintenance
- Variables
- Conditional statements
- Variable testing using the IF or SWITCH macros to provide conditional flow within template pages
- Include statements
- The FUNCTION macro, providing a mechanism for invoking C callable shared libraries and dynamic link libraries

Visual DBA Features

Visual DBA features in Ingres II 2.5 include:

- DOM windows – new design and features
- SQL/test – new design and features
- Star management
- Full Replicator management
- ICE management
- Enterprise access
- Miscellaneous new features

Replicator Enhancements

Enhancements have been made to the internal performance that concern the generic Replicator Server and increased replicator concurrency.

Generic Replicator Server

Ingres II 2.5 introduces a generic Replicator Server that is functionally identical to the custom repserver built by the user in previous releases. The generic Replicator Server can automatically handle any table that is configured by RepManager or Visual DBA without the need to compile or link a custom executable.

Increased Replicator Concurrency

In previous releases, updates to the Replicator shadow, archive, and input queue tables performed by the DBMS as a result of a replicated user update would use the same isolation level as the original update (serializable by default). This isolation level is unnecessary, since the unique key value in the base table must have already been locked for the update to take place. In this release, the isolation level has been decreased to read committed, allowing for improved concurrency of replicated updates.

OpenAPI Enhancements

Environment handles were added to OpenAPI. Environment handles allow greater application control over configuration of the OpenAPI runtime environment. OpenAPI version 2 was added to indicate support for environment handles.

Extensions were made to the following OpenAPI functions to support environment handles:

- `Ilapi_initialize()` to allocate environment handles
- `Ilapi_connect()` to open connections using environment settings

Three new environment handle associated functions were added:

- `Ilapi_setEnvParam()` for setting environment configuration parameters
- `Ilapi_formatData()` for converting data values relative to environment settings
- `Ilapi_releaseEnv()` to release environment handle resources

Also added is the function `Ilapi_abort()` to forcefully shut down connections under error conditions.

Features Introduced in Ingres II 2.0

This appendix describes the features and enhancements introduced in Ingres II 2.0, including:

- Variable page sizes
- Larger tuple support
- Distributed multi-cache management
- Enhanced performance of locking/logging and buffer manager
- Interval based deadlock detection
- Row level locking and transaction isolation levels
- Alter table support
- Async I/O support
- Parallel backup and restore
- Fast load support
- R-tree support
- Spatial data types and operators
- Statement level rules
- Temporary tables as procedure parameters
- Other optimizer enhancements
- Operating system thread support
- Table cache priorities
- Transaction Access mode
- Soundex function
- Ingres Star features
- Ingres Net features
- Ingres OpenAPI enhancements
- Ingres ICE features
- Visual DBA features
- Server-based replication

Variable Page Size

Ingres II 2.0 introduced support for multiple page sizes. The supported page sizes are 2 KB, 4 KB, 8 KB, 16 KB, 32 KB, and 64 KB. The following semantic change has been made to DDL to support the variable page sizes. A `PAGE_SIZE` clause has been added to create table, create index, modify, and declare global SQL statements:

```
CREATE TABLE ..... WITH PAGE_SIZE=n;  
CREATE INDEX ..... WITH PAGE_SIZE=n;  
MODIFY TABLE ..... WITH PAGE_SIZE=n;  
DECLARE GLOBAL .... WITH PAGE_SIZE=n;
```

Valid page sizes are 2048, 4096, 8192, 16384, 32768, and 65536.

If the `WITH PAGE_SIZE` clause is omitted, Ingres uses the configuration parameter `default_page_size` to determine the page size. For compatibility reasons, `default_page_size` defaults to 2048 (2 KB). The different page sizes are not automatically available. The DBA must configure the buffer cache of the installation to a specific page size; otherwise, an error occurs. For new buffer manager configuration parameters, IMA changes, and standard catalog interface changes, see the *Database Administrator Guide*.

New Page Format for Larger Page Size

Ingres II 2.0 uses a different physical page format for larger page sizes. The page headers have been modified for larger page sizes to provide 64-bit TID support in the future. Additional features such as larger tuples, alter table support, and row level locking utilize the new page formats and are available only for larger page sizes. With larger pages, for each tuple, a 24-byte tuple header is stored and the line table entry for larger page sizes has been increased to 4 bytes. See the *Database Administrator Guide* and the *System Administrator Guide*.

Larger Tuple Support

Ingres II 2.0 supports larger tuples (up to 32 KB) with larger page sizes (64 KB). The availability of larger tuples is dependent on the availability of larger pages, for example, through buffer manager configuration. With larger pages, maximum tuple length of the installation can be configured by the configuration parameter `max_tuple_length`. If `max_tuple_length` is set to a larger value, the DBMS Server, Star Server, and Recovery Server require more memory. For compatibility, the `max_tuple_length` parameter defaults to 2008 bytes. A value between 1 and 2008 is not recommended. If the `max_tuple_length` is set to 0, `max_tuple_length` is dependent on the availability of the larger buffer manager.

Distributed Multi-Cache Management

Ingres II 2.0 supports Distributed Multi-Cache Management. You can use multiple servers using private buffer caches and fast-commit. The Distributed Multi-Cache Management exploits the Cluster Technology using a shared disk. An installation can use either shared cache or distributed multi-cache. No mixture is permitted. Unlike non-fast-commit servers, Distributed Multi-Cache Management allows DBAs to configure the servers using fast-commit. Distributed Multi-Cache Management should not be used in a single CPU machine or when a single (sole) server is used. A new configuration parameter, `dmcm`, has been added to configure Distributed Multi-Cache Management.

Enhanced Performance of Locking/Logging and Buffer Manager

Ingres II 2.0 is optimized for performance using multiple mutexes to protect the internal critical data structures for the DBMS. Previous Ingres releases used single mutexes to protect the locking, logging, and shared buffer manager systems. The Ingres II 2.0 changes eliminate concurrency bottlenecks in these systems. Changes in the log records render all previous checkpoints invalid. After you upgrade to Ingres II 2.0, we recommend that you checkpoint all your databases for further recovery. Old checkpoint files from CA-Ingres 6.4 and CA-Ingres 1.2 cannot be rolled forward in Ingres II 2.0.

Interval Based Deadlock Detection

Ingres II 2.0 uses an interval-based deadlock mechanism to detect deadlocks. This also optimizes the contention within the Ingres lock manager and reduces CPU usage by the Ingres DBMS. Prior to Ingres II 2.0, all locks caused a deadlock search when blocked. In Ingres II 2.0, we check for deadlocks at a regular interval. A new special thread in the Recovery Server has been created to do this Interval Based Deadlock Detection. The interval for deadlock detection is controlled dynamically by the Ingres II 2.0 server based on the server utilization.

Row Level Locking and Transaction Isolation Levels

Ingres II 2.0 supports row locking for tables created with page size greater than 2 KB. The user may request row locking in cases where page locking causes unnecessary contention.

Row locking is only available to fast-commit servers. Row locking cannot be used with distributed multi-cache (DMCM) servers.

The set lockmode statement allows you to set the lock granularity, overriding the default locking strategy selected by the optimizer. The syntax for the set lockmode statement has been changed to support level=row.

set lockmode session | on *table*
where level = row | page | table | session | system

Because it is difficult for an optimizer to determine when to use row locking, the user must decide with the set lockmode statement. The default lock granularity is page.

Row locks and intended page locks are counted per transaction, and escalation to TABLE locking occurs if this count reaches the maximum locks per transaction. Row locks and intended page locks are not counted for the MAXLOCKS per query. Ingres II 2.0 supports four SQL-92 transaction isolation levels. The supported levels are serializable, repeatable read, read committed, and read uncommitted.

The new set transaction statement lets you specify the isolation level for a transaction. For details, see the *SQL Reference Guide*.

The read uncommitted isolation level does not take read-locks, and allows dirty read. The read committed isolation level takes read-locks, and then immediately releases them; this prevents dirty read, but allows unrepeatable read.

The repeatable read isolation level takes and holds read-locks on all data actually read by a query, preventing unrepeatable read, but permitting phantom read.

The serializable isolation level takes and holds read-locks on all rows potentially accessed by a where clause, preventing phantom read. The serializable isolation level is equivalent to the default behavior in previous releases.

Alter Table Support

In Ingres II 2.0, the alter table statement supports add and drop column functionality. For details on the semantic changes, see the *SQL Reference Guide*.

In this release, NOT NULL in the null_clause and WITH DEFAULT in the default_clause is not supported. Alter Table add/drop column is only available to page sizes larger than 2048 KB. Ingres II 2.0 uses a versioning technique to support the above SQL-92 compliant alter table statement. For performance reasons, the statement does not update user data. The newly added column defaults to NULL value for the existing rows. Note that alter table statement does not reclaim space for the deleted columns. You can modify the table to reclaim the space for the deleted columns.

Async I/O Support

On operating systems that support asynchronous I/O, Ingres allows the use of either I/O slaves or in-process asynchronous I/O, which may provide a performance improvement for those systems.

The basic tenet for a DBMS (and RCP) server is that disk I/O should never cause the server to block while there is other work pending. To achieve this, Ingres has passed all disk I/O requests to separate processes (called *I/O slaves*), up to a limit of 30 per server, to do the actual disk operations. The slave processes perform the task synchronously and notify the server when it is completed. Each I/O operation using I/O slaves involves, among other things, a number of context switches from server to slave and back again. Context switching between processes is an expensive task when compared to context switching within a process. Performing the same non-blocking I/O within the server process provides a more efficient way to achieve non-blocking I/O.

Ingres II 2.0 provides a new per-server configuration parameter called `async_io` that can be set in CBF to enable the in-process asynchronous I/O and disable I/O slaves. In a multiple-server installation, each server can be configured separately to use either slaves or asynchronous I/O.

For operating systems that allow OS threading (discussed later), neither slaves nor async I/O is optimal. The best choice on OS-thread platforms is in-thread blocking I/O.

Parallel Backup and Restore

Parallel backup and restore enhancements in Ingres II 2.0 include:

- Parallel checkpointing to disk
- Parallel checkpointing to tape
- Parallel rollforwarddb from disk
- Parallel rollforwarddb from tape

Parallel Checkpointing to Disk

To checkpoint a multi-location database to disk in parallel, use the `#m` flag followed by the number of parallel checkpoints to be run:

```
ckpdb '#m2' dbname
```

This saves two data locations at a time to the `II_CHECKPOINT` location.

Parallel Checkpointing to Tape

To checkpoint a multi-location database to tape in parallel, list the devices to be used with the `-m` flag:

```
ckpdb -m/dev/rmt/0m,/dev/rmt/1m dbname
```

This saves one location per tape. The first location is stored on device 0m. The second location is stored on device 1m. The third location is stored on whichever device is done first. The remaining locations are stored on the next free device. The operator is prompted to insert a new tape for each location.

Parallel Rollforwarddb from Disk

To roll forward a multi-location database to disk in parallel, use the `#m` flag followed by the number of parallel restores to be run:

```
rollforward '#m2' dbname
```

This restores two data locations at a time from the `II_CHECKPOINT` location.

Parallel Rollforwarddb from Tape

To roll forward a multi-location database from tape in parallel, list the devices to be used with the `-m` flag:

```
rollforward +c -m/dev/rmt/0m,/dev/rmt/1m dbname
```

The first location is restored from device 0m. The second location is restored from device 1m. The third location is restored from whichever device is finished first. The remaining locations are restored from the next free device. The operator is instructed to insert the numbered tape into the free device.

Fast Load Support

Ingres II 2.0 supports a new fast bulk loading of data into a single table in a single database; the fastload command can load binary formatted files into database tables. In contrast to the copy statement, the fastload command is run from the command line and uses Ingres low-level record management functions directly to load the data. It does so without the use of a connection to an Ingres DBMS, and thus avoids passing data through communications channels on the system. The user's process reads the input file and writes to the table in the database. The loading of some complex data types and the use of some table structures are not supported in all situations. For details, see the *Database Administrator Guide*.

R-tree Support: A Spatial Index for Ingres II 2.0

This is a new access method for secondary indexes on ordered pair data. This is directed at spatial data types, but it can be extended to user-defined types. R-tree indexes can be created for any base table type: B-tree, hash, heap, and isam, using the create index command with structure=rtree syntax.

This access method provides lightning lookup to solve range queries and spatial joins for large numbers of objects. Record pointers are stored in spatial sequence (that is, consecutive objects in the index represent objects that are physically close). An R-tree index participates in the usual query optimization and is used with the operators inside, intersects, and overlaps.

Spatial Data Types and Operators

There are integer forms of point, box, lseg, line, polygon, and circle. Up to 249 ipoints can be used in an iline, versus 124 points in a line. Line, iline, polygon, and ipolygon are stored in compressed variable length form when "compression=data" is defined for the table. Spatial data types can be constructed from their base components. For example, mybox = box (llpoint, urpoint).

New data types, `nbr` and `hilbert`, can be used to order spatial data. A new `overlaps` operator determines if two objects have any points in common. Spatial operators support infix notation. For example:

```
select count(*) where property_location intersects pond-area;
```

Statement Level Rules

Prior to Ingres II 2.0, row level rules support resulted in the invocation of the rules procedure for every row qualified by the triggering statement. This can entail a significant amount of overhead, particularly for auditing rules applications that may be interested in the number of rows touched rather than their actual contents.

The statement level rules feature of 2.0 allows users to create rules that call the corresponding procedure exactly once for each execution of the triggering statement, rather than once for each row touched. The same information is still made available to the procedure as with row-level rules. For each qualifying row touched by the triggering statement, a row is inserted into an internal temporary table. This row contains the data from the parameter list of the rule definition's `execute procedure` statement. The entire temporary table is then passed to the rule procedure for processing (see [Temporary Tables as Procedure Parameters](#) in this appendix).

Temporary Tables as Procedure Parameters

Ingres database procedures previously had limited ability to process bulk information. This is largely the result of the restriction requiring that scalar parameters be passed between the caller and the procedure. Release 2.0 permits a global temporary table to be passed as a parameter to an Ingres database procedure. The temporary table can contain as many rows as desired upon entry to the procedure, permitting the procedure statements to process all the data with a single call. Likewise, the temporary table can be empty on entry to the procedure, to be filled by retrieval statements inside the procedure. The declaration of the temporary table inside the procedure allows it to be treated as any other Ingres table. It can be referenced in the "from" clause of a select or update statement and can be the target of an insert, delete, or update statement within the procedure. The potential effect of the feature is to reduce the number of procedure calls required to process a given quantity of data.

Other Optimizer Enhancements

Several changes were made to the optimizer in Ingres II 2.0 to improve the quality of the generated query plans. The changes include the following:

- Reduction in size of generated query plans
- Improvement of join cardinality estimates
- Improvement of selectivity estimates for <> (not equals) restriction predicates

Following similar improvements in Ingres 1.2, changes were made to various internal query plan structures to further reduce the size of query plans generated by Release 2.0. The combined effect of the 1.2 and 2.0 changes reduces typical query plans by 40 percent to 60 percent. This frees QSF cache for use by more queries, resulting in significantly improved cache utilization.

A new statistic and new formulas have been introduced into Release 2.0 which should result in more accurate estimates of the number of rows produced by the join of two tables in an Ingres query. Since row estimates are a major component of the algorithms used to generate query plans, these enhancements should result in improved query optimization. The new statistic is a “per-cell” repetition factor. It reflects the average number of rows per distinct column value for EACH cell of a histogram, thus allowing more accurate join estimates. The new statistic is automatically computed by Release 2.0 `optimizedb` requests, and is displayed by Release 2.0 `statdump` requests. Moreover, Ingres II 2.0 uses the default for prior release histograms, so that no statistics upgrade is required.

Ingres has always used histogram values to estimate the number of rows qualified by `=`, `<`, `<=`, `>=`, and `>` restriction predicates. As with the join cardinality estimates, these accurate row counts allow Ingres to build optimal query plans. Release 2.0 introduces histogram-based selectivity estimates for `<>` (not equals) restriction predicates, as well. This allows Ingres II 2.0 to generate even better query plans.

Operating System Thread Support

A major enhancement to Ingres on operating systems that support operating-system threads is greater support for symmetrical multi-processor (SMP) machines. The database server takes full advantage of the ability of the operating system to balance processing time among all the available CPUs. By mapping database tasks to operating-system threads, individual sessions are dynamically balanced among available CPUs, providing greater throughput and scalability.

OS threading reduces or eliminates the need to run multiple DBMS servers in an installation. Because the OS instead of the DBMS is doing the scheduling, load balancing and Ingres administration is greatly simplified. OS threading also allows individual threads to block on I/O, eliminating the need for I/O slaves or complex async I/O schemes.

Table Cache Priorities

Database page cache priorities are normally assigned and managed by algorithms within the Ingres buffer manager. This addition allows tables to be assigned fixed priorities, which reduces the page replacement rate for those tables in a properly configured cache. For the syntax for assigning fixed cache priority, see the *SQL Reference Guide*.

Transaction Access Mode

Support is included for the access mode of a transaction. For syntax information, see the *SQL Reference Guide*.

Soundex Function

Ingres II 2.0 supports a new string function—SOUNDEX. The results of SOUNDEX are a char(4), and similar sounding words have the same SOUNDEX value. This is particularly useful when searching for names. For example:

```
SELECT fname, lname, addr1, addr2 from customer_table
WHERE SOUNDEX(fname) = SOUNDEX('kathy');
```

Note: The SOUNDEX function is case-insensitive.

Ingres Star Features

Ingres II 2.0 Star Server supports variable page size and larger tuples as well as connectivity with previous CA-Ingres releases. To support variable page size, the standard catalog interface has been changed to include page sizes and other catalog changes. For a description of the new standard catalog interfaces, see the *Database Administrator Guide* and the *System Administrator Guide*.

Ingres Net Features

New features for Ingres Net include the following.

Protocol Bridge Support

Ingres Protocol Bridge resolves the problem of connecting an Ingres client on one network to a server on a different type of network. Under the current architecture, a client and server must be able to communicate over the same network protocol (such as TCP/IP or SNA_LU62). The protocol bridge will “bridge” a client using one network protocol to a server using another. For example, a PC on a TCP/IP network could communicate through the protocol bridge to an EDBC gateway (DB2, IMS, CA-Datcom/DB, and so on) on an SNA network.

The protocol bridge provides communication capability between incoming connections and outgoing connections using a **different underlying protocol**. It listens for and accepts incoming connection requests and establishes corresponding connections to a local or remote communication server, allows bi-directional data transfer over the established connections, and terminates connections in an orderly way.

The protocol bridge does not provide any security checking when passing the messages through; security is still handled as usual on the server.

For information on installing, starting and stopping, monitoring, and tracing the Protocol Bridge Server, see the *System Administrator Guide*.

Data-Stream Compression Support

Ingres II 2.0 supports data-stream compression for variable-length data types, such as varchar and byte varying. Previously, when a client performed a COPY FROM or SELECT operation, the DBMS would send varchars at their maximum lengths, even if only a small fraction of the varchars contained usable data. In Ingres II 2.0, the DBMS sends only the usable data. This is true for both local and network connections.

Database administrators may disable data-stream compression by invoking CBF, and setting the vch_compression parameter of the DBMS to OFF. This disables compression when the DBMS is re-started. Individual clients may disable compression by setting the environment variable II_VCH_COMPRESS_ON to N. Setting II_VCH_COMPRESS_ON to Y, or unsetting II_VCH_COMPRESS_ON, re-invokes data-stream compression only if the vch_compression parameter of the DBMS is ON.

SNA Duplex Support

In previous releases of Ingres/Net, a single SNA LU 6.2 conversation was used to support each client/server SNA connection. Since LU 6.2 conversations are inherently half-duplex, and Ingres transactions cannot be guaranteed to be, a large amount of overhead was created in order to maintain the flexible Ingres client/server dialog. This was a problem only with the SNA LU 6.2 protocol driver since the other protocols supported by Ingres/Net use full duplex protocol.

In this release of Ingres/Net, the overhead of maintaining the half-duplex nature of SNA LU 6.2 has been eliminated by using two conversations per client; one conversation stays in send state, the other in receive state. There are two important consequences of this implementation:

- Response time shows an approximate 50% improvement as the result of avoiding all request-to-send and prepare-to-receive calls.
- Only half the number of clients can be supported.

This feature can be controlled by a new environmental variable, `II_HALF_DUPLEX`. If this variable is set to 1, classic CA-Ingres/Net half-duplex operation is enabled. If this variable is undefined or set to 0, Ingres/Net 2.0 full duplex operation is enabled.

DECNet/OSI Support

On VMS systems, users may enter Phase-V-style node names, which may be greater than seven characters, for node addresses.

Ingres OpenAPI Enhancements

New features for OpenAPI include the following.

Multi-Threaded OpenAPI

Ingres II 2.0 OpenAPI supports multi-threading development and runtime environments for multi-threaded applications. Multi-threading support is platform-dependent. For more details, see the platform-specific Release Notes Supplement. There is no external change required for multi-threaded OpenAPI support.

OpenAPI Support for Autocommit

Ingres II 2.0 OpenAPI supports the SET AUTOCOMMIT statement through special autocommit transaction handles. Autocommit transactions are enabled/disabled through the new OpenAPI function, `Ilapi_autocommit()`. For details, see the Ingres II 2.0 *OpenAPI User Guide*.

Enhanced OpenAPI Support for Database Events

Ingres II 2.0 OpenAPI introduces enhanced support for database events with the `Ilapi_getEvent()` function. This function permits an application to wait for database event notification when client/server activity is idle. Previously, applications would only receive database event notifications when other non-event related client/server communications were occurring.

Ingres ICE Features

New Ingres/ICE features for Ingres II 2.0 include:

- Support for the Microsoft Internet Information Server on Windows NT through a version of Ingres/ICE written for Microsoft ISAPI (Internet Server API)
- Support for NSAPI (Netscape API)
- Connection caching for ISAPI, ADI, and NSAPI interfaces, providing improved performance
- Requests and result pages can be up to 2 GB in size, subject to available disk and memory space
- Support for SQL select statements.

The result set of the select is automatically formatted as a table and displayed on the client browser.

- Support for BLOB data

Images stored as binary database objects may be selected and displayed on the client browser.

- A new HTML macro feature

Template HTML files that contain simple macros specifying SQL statements can be created. When Ingres/ICE loads the macro file, it replaces the macros with the result of the SQL statements, allowing data from different tables and databases to be combined on a single page.

- Improved security
Ingres/ICE interoperates with the security configuration of the target Web server. The HTTP basic authentication protocol is fully supported. This means that user IDs and passwords can be passed in an encoded form across the network.
- HTML-based installation and configuration utility

Visual DBA Features

Visual DBA 2.0 differentiates between the following installations:

- Ingres 2.x installation
- CA-Ingres 1.x installation
- Local CA-Ingres/Desktop server
- Other installations

Users can connect and open simultaneously several windows displaying these installations. The ability to drag and drop tables is available between CA-Ingres 1.x, Ingres 2.x, and local CA-Ingres/Desktop server windows.

Enhancements

The new Visual DBA features for Ingres II 2.0 installations include:

- Alter table support through Visual DBA 2.0
- Support of variable page size for Create Table, Create Index, and Modify commands, as well as the space calculations
- Management of table-level unique constraints and table-level check constraints both in Create and Alter Table
- References sub-dialog of Create Table is redesigned
- Support of table-level checkpoint and rollforwarddb
- Support for CA-Ingres Replicator versions 1.0/03, 1.0/05, 1.1, or 2.0

The following new features are also available for 1.x installations:

- Management of table level unique constraints and table level check constraints in create table
- Create table as select in Visual DBA 2.0 (with the Assistant to build the select statement)

Server-based Replication

The initial data capturing of Ingres/Replicator is performed inside the Ingres DBMS. Internal DBMS functions and system threads have superseded the use of database rules and an application external to the DBMS to maintain the Replicator input and distribution queues. Any lock contention caused by previous versions of CA-Ingres Replicator has been eliminated, and Replicator data capture work is performed at a much lower level, producing more streamlined and faster performing replication.

The Ingres/Replicator product is delivered as part of the base Ingres II 2.0 release, and the configuration and installation tools are integrated into the base Ingres setup tools. See the *Getting Started* guide.

Index

4

4GL table_key type conversions, B-3

6

64 bit file support, F-10

64-bit operating systems, E-8

A

aggregate sort nodes, E-6

alter table support, G-5

ANSI/ISO constraint, F-3

applications

- install upgraded, B-35

- installing upgraded, 4-16

- issues, 1-7

- lifecycle, 1-7

- planning, 2-3

- preparation, 2-3

- preparation if migrating from 6.4, B-1

- rebuilding in Ingres on OpenVMS, A-3

archiver exit shellscript, B-4

arithmetic errors, greater sensitivity to, B-3

async I/O support, G-5

auditdb utility enhancements in 2.6, E-3

auto commit, OpenAPI support for, G-13

automated creation of location directories, E-5

AXM build. *See* OpenVMS

B

binary level support, 1-5

bit-wise operator support, F-9

buffer manager, G-3

- performance improvements, E-8

BYREF errors, greater sensitivity to, B-2

C

character data types maximum size in 2.6, E-2

character sets for Euro currency symbol, E-11

checkpoint

- template, 2-8, 3-7, B-16

- the database, B-8

compiling applications, 2-7

concurrency, Replicator, F-14

concurrent rollback, E-6

Configuration-By-Forms, 4-11, B-17

conversions, 4GL table_key type, B-3

copydb utility enhancements, E-3

D

data type changes, user-defined, B-3

database

- checkpoint, 4-15, B-35

- destroying, 4-7, B-25

- events, OpenAPI support for, G-13

- extend, 4-13, B-33
- information, record, 3-3, B-10
- moving, 2-5
- recreate, 4-13, B-32
- unload, 4-5, B-23

DBA enhancements in 2.6, E-3

decimal constant, semantics change, B-2

DECNet/OSI support, G-12

default locations, record, B-13, B-28

destroydb command, 4-7, B-25

distributed multi-cache management, G-3

Distributed Option
databases, moving, 2-6

dmf sort, F-2

dynamic write behind threads, F-5

E

enhancements
in Release 2.0, G-1
in Release 2.5, F-1
in Release 2.6, E-1

errors, arithmetic, B-3

Euro currency symbol support, E-11

extended date support, F-10

F

fast load support, G-7

FE reload script, fix, 4-14, B-34

front-end upgrade, 4-15, B-34

G

GatherWrite threads, E-4

generic Replicator Server, F-14

H

hardware
implementation, 1-5
planning, 1-5
testing, 1-5

histograms, composite, E-7

I

ICE
development environment, E-9
document cache management, F-13
enhancements in 2.0, G-13
enhancements in 2.5, F-12
enhancements in 2.6, E-9
macro language extensions, F-13
security, F-12
session management, F-13
storage management, F-13

iidbdb, clean, 4-7, B-25

image file formats, OpenROAD 4.0, 2-12

Import Assistant, E-4

infodb utility, 3-3, 4-5, B-10, B-23

ingprenv
record default locations, B-13, B-28
replaces ingprenv1, B-4

ingprenv1, B-4

Ingres 6.4
considerations for, B-1
upgrading from using unload/reload, B-19
upgrading from using upgradeb, B-5

Ingres Net. *See* Net

Ingres Star. *See* Star

installation
development, 1-5
production, 1-5
testing the upgrade, 1-5

internal performance enhancements in 2.6, E-6

interval-based deadlock detection, G-3

J

JDBC enhancements in 2.6, E-10
Journal Analyzer, E-4
journaling on by default, B-3

K

keywords, D-1
 reserved, 2-3
 single, D-2

L

language enhancements in 2.6, E-1
large cache support, F-4
large catalogs, F-10
locking system performance, E-7, G-3
logging system performance, E-8, G-3
logins, fix, B-13, B-27

M

member_aligned version, A-3
Metaschema module, 2-6
Microsoft Transaction Server support, E-6
migrating
 Ingres II 2.0 to AXM on OpenVMS, A-1
migrating on OpenVMS
 building COBOL applications, A-5
 building member_aligned against Ingres, A-3
 C applications, building, A-4
 installation issues, A-2
 installing Ingres, A-1
 rebuilding applications, A-3
 schema checking, A-3
multi-threaded OpenAPI, G-12

N

Net, 1-5, G-11
 data-stream compression support, G-11
 DECNet/OSI support, G-12
 enhancements in 2.5, F-12
 protocol bridge support, G-11
 setup, 4-11, B-17, B-31
 SNA duplex support, G-12
netutil, 4-11, B-31
new features
 in Release 2.0, G-1
 in Release 2.5, F-1
 in Release 2.6, E-1

O

ODBC driver, E-9
OpenAPI, G-12
 multi-threaded, G-12
 support for auto commit, G-13
 support for database events, G-13
OpenROAD 4.0 image file formats, 2-12
OpenVMS
 migrating to Ingres II 2.0 AXM on, A-1
OpenVMS requirements, A-1
operating system integration in 2.6, E-8
operating system thread support, G-9
optimizeddb, F-6
optimizer, G-9
 reapply statistics, 4-15, B-18, B-35
 support for hash joins, E-7

P

page format, new, G-2
parallel
 backup and restore, G-6
 checkpointing to disk, G-6
 checkpointing to tape, G-6
 rollforwarddb from disk, G-6
 rollforwarddb from tape, G-6

- sort techniques, F-3
- parameters, UNIX kernel, 2-10
- partitioned transaction log file, F-5
- preallocated rsb/lkbs, E-7

Q

- qef sort, F-2
- query optimization, F-11

R

- raw location support, E-4
- read-only database support, F-6
- record output configuration, B-25
- reload upgrade, 4-1, B-19
- reload.ing, 4-15, B-34
- Remote Command Server, 3-2, 4-4, E-6
- Replicator enhancements
 - generic server, F-14
 - increased concurrency, F-14
- Report-Writer
 - runtime parameter errors, 2-4
 - syntax change, 2-4
- reserved words, A-3, D-1
 - conflicts, 2-6
 - new, 2-3
- row
 - level locking, G-4
 - locking for system catalogs, F-10
 - producing procedures in 2.6, E-1
- R-tree support, G-7

S

- scripts
 - fix FE reload, 4-14, B-34
- search path, shared library, 2-9

- server-based replication, G-15
- shared library search path, 2-9
- shellscripts
 - archiver exit, B-4
 - for system monitoring, 2-8
- showrcp command, B-25
- shut down Ingres, 3-2, 4-4, B-4, B-8, B-22
- site modifications
 - preserve, 3-4, 4-8, B-12, B-26
 - restore, 3-7, 4-11, B-16, B-30
- sort enhancements, F-2
- soundex function, G-10
- spatial data types and operators, G-7
- SQL functionality, F-7
 - bit-wise operator support, F-9
- Star. *See also* Distributed Option
 - features in 2.0, G-10
 - features in 2.5, F-11
- start Ingres, 3-7, 4-12, B-16, B-31
- startup, 3-7, 4-12, B-4, B-16, B-31
 - disable, 4-7, B-26
- statdump command, 4-5, B-9, B-23
- statement level rules, G-8
- storage structures, reapply, B-18
- syntax, Report-Writer, 2-4
- system administration
 - backup, 2-9
 - practice upgrade, 2-11
 - preparation, 2-8
 - restore, 2-9
 - testing, 2-11
- system backup, 3-2, 4-4, B-8, B-22
- system monitoring shellscripts, 2-8
- system_maintained column name, 2-7

T

- table cache priorities, G-10
- temporary tables as procedure parameters, G-8

- testing, 2-7
- thread implementation on Linux in 2.6, E-9
- transaction
 - access mode, G-10
 - isolation levels, G-4
 - log size, B-5
- tuple size, larger, G-2

U

- Unicode support, E-10
- UNIX kernel parameters, 2-10
- unload directory, create, 4-2, B-6, B-21
- unload upgrade, 4-1, B-19
- unload/reload procedure, 4-2, B-20
 - back up system, 4-4, B-22
 - check for obsolete users, 4-3, B-21
 - checkpoint database, 4-15, B-22, B-35
 - clean iidbdb, 4-7, B-25
 - configure Ingres, B-30
 - create unload directory, 4-2, B-21
 - create work location, B-28
 - destroy database, 4-7, B-25
 - disable Ingres startup, 4-7, B-26
 - extend database, 4-13, B-33
 - fix FE reload script, 4-14, B-34
 - fix logins, B-27
 - install Ingres, 4-9, B-29
 - install upgraded applications, 4-16, B-35
 - optional checkpoints, 4-3
 - preserve site modifications, 4-8, B-26
 - print optimizer statistics, 4-5, B-23
 - reapply optimizer statistics, 4-15, B-35
 - record default locations, B-28
 - record infodb output, 4-5, B-23
 - record Ingres configuration, B-25
 - recreate database, 4-13, B-32
 - reload database, 4-15, B-34
 - restore site modifications, 4-11, B-30
 - run unloaddb, 4-2, B-21
 - set up Net, 4-11, B-31
 - shut down Ingres, 4-4, B-22
 - start Ingres, 4-12, B-31
 - unload database, 4-5, B-23
 - upgrade front-end catalogs, 4-15, B-34
 - when to use, 1-2
- unloaddb command, 2-5, 2-6, 4-2, 4-5, B-5, B-6, B-21

- output, B-7
- update mode locking, F-11
- UPDATE...FROM semantics change, B-1
- upgrade
 - applications, 1-7, 4-16, B-35
 - hardware issues, 1-5
 - planning, 1-1
 - using unload/reload procedure, 4-1, B-19
 - using upgradedb procedure, 3-1, B-5
- upgradedb procedure
 - application upgrade, 3-8, B-19
 - back up system, 3-2, B-8
 - check for obsolete users, B-7
 - checkpoint the database, 3-3, 3-8, B-8, B-11, B-19
 - clean iidbdb database, B-11
 - clean up Ingres 6.4, B-14
 - configure Ingres, B-17
 - create unload directory, B-6
 - create work location, B-14
 - disable Ingres startup, B-12
 - disable user access, 3-1, B-8
 - edit unloaddb output, B-7
 - fix logins, B-13
 - install Ingres, 3-5, B-15
 - preserve site modifications, 3-4, B-12
 - print optimizer statistics, B-9
 - reapply optimizer statistics, B-18
 - reapply storage structures, B-18
 - record database information, 3-3, B-10
 - record default locations, B-13
 - record Ingres configuration, B-11
 - recreate objects, B-18
 - remove non-table objects, B-9
 - restore site modifications, 3-7, B-16
 - run unloaddb, B-6
 - run upgradedb utility, 3-7, B-16
 - set up Net, B-17
 - shut down Ingres, 3-2, B-8
 - start Ingres, 3-7, B-16
 - when to use, 1-2
- user access, disable, 3-1, B-8, B-22
- user check, 4-3, B-7, B-21
- usermod utility, E-3

V

- value locking protocol, F-11

variable page size, G-2

Visual DBA, F-14, G-14

VMSinstal, running, A-2

X

xml import/export utility, E-4

W

work location, create, B-14, B-28