Ingres® 2006 Release 2

Ingres Star User Guide

INGRES®

February 2007

Contents

Chapter 1: Introducing Ingres Star

This guide describes how to implement and use Ingres® Star and its related facilities.

Ingres Star

Ingres Ingres Star (formerly Ingres Distributed Option) is a distributed data manager that adds to Ingres a distributed relational database management system capability, which includes distributed access, storage, and processing. You can include multiple hardware and operating systems (mainframe, mid-range, and desktop) and database management systems in this distributed system.

With Ingres Star, you can combine many separate databases into a single view of your data, which you can access just as you would any single, local database. If you install Ingres Net and the Enterprise Access products along with Ingres Star, you get transparent, simultaneous access across multiple nodes, hardware platforms, and software configurations by means of multiple network communication protocols.

Audience

This guide is addressed to two levels of Ingres users:

- For end users, this guide explains the concepts of distributed database processing and how Ingres Star provides access to data in multiple local Ingres databases, and to non-Ingres databases through the use of Enterprise Access products.
- For the system administrator and the database administrator, this guide details how to use and maintain Ingres Star and the catalogs. Individual product guides address installation and maintenance of the Enterprise Access products.

Query Languages

This guide uses the industry standard query language, SQL. QUEL is not supported in Ingres Star.

Installation Considerations

The type and location of the databases you wish to access determine which Ingres products you must install.

- For local data access:

  Ingres by itself works with one database at a time on your own computer. Therefore, to access a single database on a single computer, just install Ingres on your local computer.

- For remote data access:

  When your computer is part of a network, Ingres Net allows you to access a single database stored on another computer from your local computer. Therefore, to access a database on a remote computer, you must install Ingres Net both on your local computer and the remote computer.

  For information, see the *System Administrator Guide*. For information about using Ingres Net with Ingres Star, see Setup for Remote Database Access (see page 11).

- For heterogeneous data access:

  The Enterprise Access products allow you to access data stored in non-Ingres databases. Therefore, to access a non-Ingres database, install the appropriate Enterprise Access for that database type. See the guides that are specific to that Enterprise Access.

  Note: If the non-Ingres database is on a remote computer, you must also install Ingres Net.

- For distributed data access:

  Ingres Star gives you access to multiple databases simultaneously. You need not even know where the data you need resides. The other databases and their configurations are invisible to you. You work transparently with Ingres Star distributed database as if it were a single local Ingres database on your own computer.

  Combining Ingres Star with Ingres Net gives you access to remote as well as local databases, no matter where those databases are stored on your network.

  Combining Ingres Star with Enterprise Access products allows you to access both non-Ingres and Ingres databases, or even several different types of non-Ingres databases. For example, you can create a distributed database that contains Ingres, DB2, and IMS data. By joining them together in a single distributed database, you can perform complex operations such as joins and subselects between DB2 and IMS data that would be very difficult without the use of Ingres Star.

  Combining Ingres Star with Ingres Net and Enterprise Access or EDBC products allows you to access both Ingres and non-Ingres databases simultaneously, anywhere on your network.

Setup for Remote Database Access

To work with an Ingres database stored on another computer, the computer on which you are entering commands and the computer that contains the database must be connected by a network. (The computer on which you are entering commands is called the *local node* and the computer that contains the database is called the *remote node*.)

To work on an Ingres database on a remote node, the following setup is required:

- Ingres Net must be installed on your local node and on each remote node.
- Either you or the system administrator must use the netutil or ingnet utility to define each remote node that contains a database that you want to access. Defining a remote node gives it a *vnode* (virtual) name. From then on, you need only specify the *vnode* name of the remote node and the name of the database. Net handles the details of gaining access to the remote node.
- Ingres Star users must be authorized on the local node as well as each remote node they wish to access, unless installation passwords are used. This authorization is also done through the netutil or ingnet utility. Use netutil or ingnet on your local node to define your login authorization to your account on the remote node. (Alternatively, the system administrator must define your access to a public guest account on that remote node).

  Note: You need to define authorization to your account on a remote node only once. Your authorization remains until you delete it.

  If installation passwords are used, the installation password of a remote node must be entered with netutil or ingnet.

For additional details, see Define User Authorization to Nodes on the Network (see page 29).

Command Syntax for Accessing a Distributed Database

The syntax for accessing a distributed database by means of Ingres Star is:

IngresCommand [*vnode*::]*distdbname*/*server_class*

IngresCommand

Is an Ingres command.

vnode

Indicates the name of your distributed database that resides on a remote node.

/server_class

Indicates the server class.

Always use **/star**, which signifies that you are accessing a distributed database through the Star Server. This requirement applies even for accessing Ingres Enterprise Access databases through Ingres Star.

Examples: Accessing Distributed Databases

To open the Ingres Menu and access a distributed database on your local node, type the following command:

```
ingmenu distdbname/star
```

The /star parameter signifies the Star server class.

To open the Ingres Menu and access a distributed database on a remote node whose vnode name is paris, type the following command:

```
ingmenu paris::distdbname/star
```

In this case, you must specify both the vnode name of the remote node and the Star server class.

Whether the distributed database is on your local node or on a remote node, once you have accessed the database by means of Ingres Star, you can work on the database as if it were a collection of tables in a single Ingres database.

Chapter 2: Understanding Ingres Star Architecture

This chapter explains the overall system architecture and components that comprise the Ingres Distributed Database Option.

Distributed Database

Ingres Star lets you access data from multiple databases, linking the information together so that it seems to be in a single database. This single database is referred to in the rest of this guide as the distributed database.

A distributed database does not reside physically on a disk. The actual data is contained in the multiple databases accessed by Ingres Star. These databases are referred to in the rest of this guide as local databases.

A DBMS Server must be running on each computer that has a database you are accessing.

Setting up a distributed database requires you to:

- Create an empty database
- Register tables
- Populate the database with the tables
- Understand Ingres Star catalogs, object types, and components

Creation of a Distributed Database

To create a distributed database, use the Ingres createdb command described in Createdb Command (see page 44). This command creates an empty distributed database.

Registration of Distributed Database Tables

After you create a distributed database, you must decide which tables will make up the distributed database. These tables can be entire local databases or a selection of tables from local databases. They are incorporated into the distributed database by registering them.

Tip: Registration simply describes the logical connection (link) between an object in the distributed database and an object in a local database. A registration is not an object in the same way that a table, view, or index is an object.

When you issue a query to Ingres Star, it generates sub-queries to the local databases, mapping table and column names occurring in the query into their corresponding local table and column names. Ingres Star cannot recognize local table or column names unless they have been registered in the distributed database.

Population of the Distributed Database

You can populate the distributed database with tables in three ways:

- By registering tables that exist in local databases. Register the names of existing tables in your distributed database using the register as link statement.
- By creating new tables at the Ingres Star level, which automatically registers them in your distributed database. When you are connected to the Star Server and create a new table with the create table statement, you create a table locally and register its name in your distributed database.
- By creating new tables in local databases at the local level and then registering these new tables in your distributed database.

For discussions of the register and create commands, see the chapter "Maintaining a Distributed Database with StarView."

Catalogs

When you create a distributed database, Ingres Star creates catalogs to coordinate access to all the data. The catalogs are a collection of tables that store information about the distributed database. They describe every local database that is a component of the distributed database, as well as the distributed database's objects. For a complete description of the catalogs, see the chapter "Ingres Star Catalogs" and the appendix "Standard Catalog Interface."

Object Types

Although a distributed database connects to data stored in local databases, it contains its own set of objects. These objects have the same types as those of a local database:

- Tables
- Views
- Procedures
- Indexes

The above description for registering tables includes the registration of views. You can also register database procedures. Indexes are registered automatically when you register the tables they belong to.

Ingres Star Components

In Ingres Star, a distributed database is defined by the following components:

- A Coordinator Database (CDB)

  A coordinator database contains the catalogs that the Star Server uses to keep track of distributed objects. When a user requires information, the Star Server accesses the coordinator database and associated local databases through the local DBMS Server to get the information. (Although the coordinator database is an Ingres database and can be accessed just like any other database, it is intended to be used solely by the Star Server, so it is advisable to access the coordinator database through Ingres Star only.)

- Entries in installation definition catalogs (iidbdb information)

  These entries contain information that define each distributed database and coordinator database in the installation, and show which coordinator database is associated with which distributed database. They also contain configuration information that allows Ingres Star to efficiently process queries.

- Optional user-specified links to data in other databases

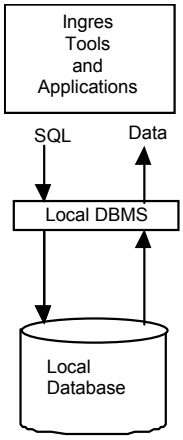
  Optional links to other data are generated by the Register as Link Statement (see page 46). When you register existing data to a distributed database, information on the location of the data is added to the Ingres Star-specific catalogs in the coordinator database.

Security

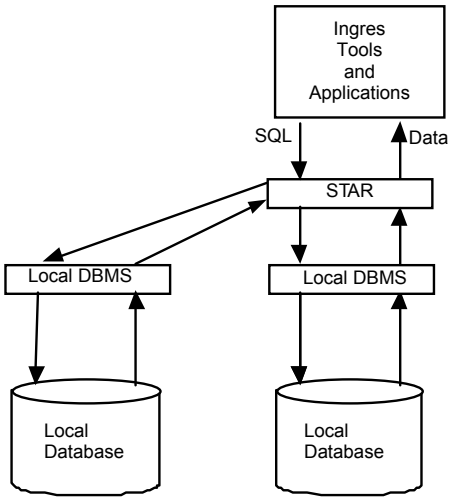
You can register tables and views that are not owned by you. However, even though you may have registered a table or view in your distributed database, you still must have the necessary permissions to access those tables and views before you can access them through your distributed database.

System Architecture

The following diagram shows a simple database access in Ingres:

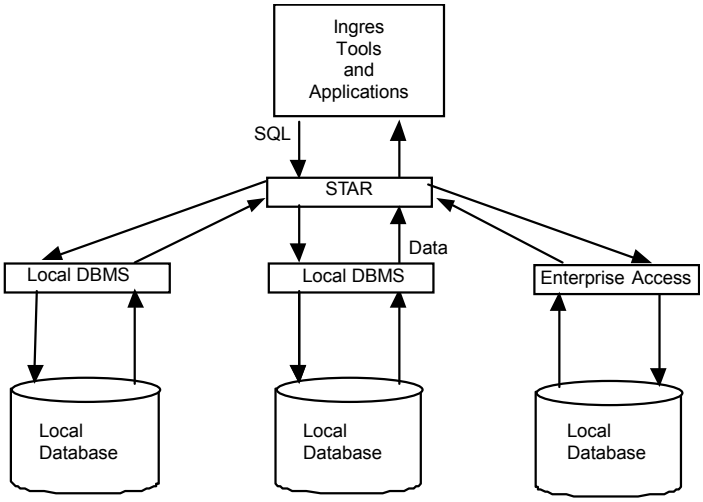


The following diagram shows how Ingres Star accesses data:



The Star Server receives requests for data from multiple clients and passes those requests for data to a local DBMS Server. (The Star Server never directly accesses data in a database.) The local DBMS servers can support simultaneous user access to local databases and queries from the Star Server. The local DBMS accesses its database and returns the results. The Star Server returns the results of the distributed query to the application.

Star can also access non-Ingres databases through Enterprise Access products. The following diagram shows how Ingres Star accesses data on two Ingres databases and one non-Ingres database linked by means of an Enterprise Access product:



Configuration Access Examples

The following examples illustrate various Ingres Star configurations and show:

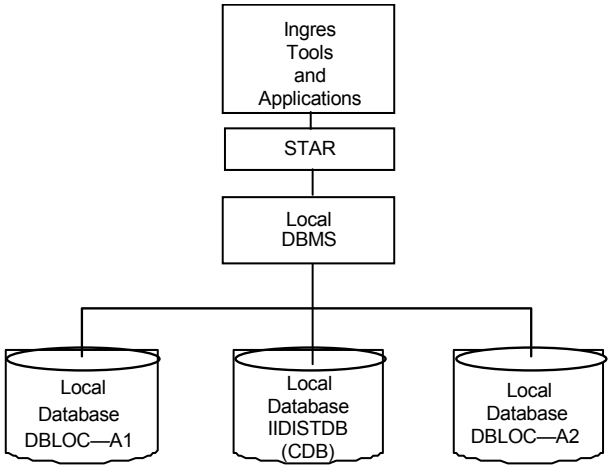
- Components of a distributed database, including Ingres Net, and the communication between the components
- Syntax for entering a distributed database session (which may differ according to the configuration)
- Definition of distributed objects and how this reflects the underlying configuration

In the examples that follow, the name of the distributed database is DISTDB.

Example: Single Node Configuration

The following diagram shows the client and the complete Ingres Star configuration on one node (Node A) and in one installation. The distributed database DISTDB consists of two local databases, DBLOC_A1 and DBLOC_A2, as well as the coordinator database IIDISTDB. As shown in the following diagram, Ingres Star's access to the local databases as well as to its catalog information in the coordinator database is through a local DBMS:

Node A



To invoke the Terminal Monitor on the distributed database, issue the following command at the operating system prompt:

```
sql distdb/star
```

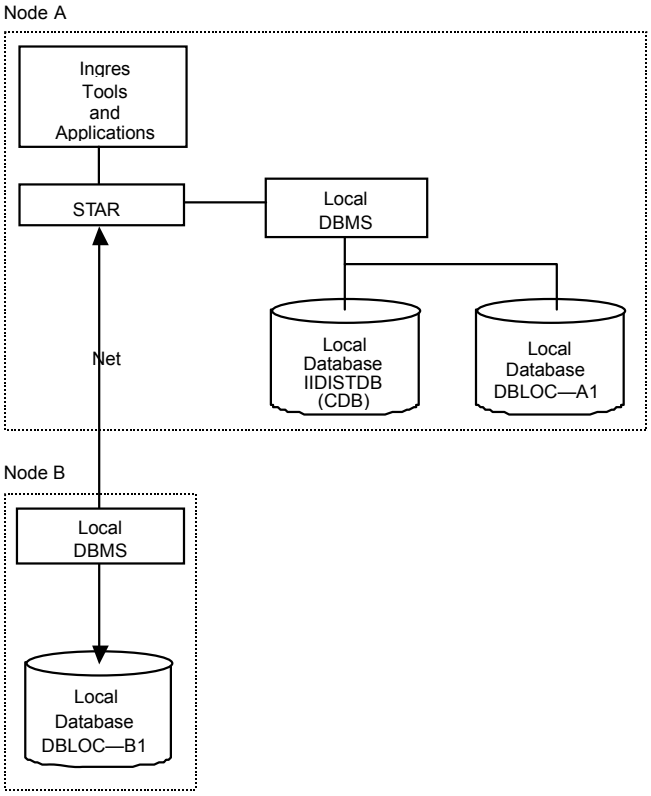
To incorporate TABLE1 in database DBLOC_A1 into the distributed database, issue the following statement in an application or an SQL session:

```
register table table1 as link
  with node = node_a,
  database = dbloc_a1;
```

When a Ingres Star user requests data from TABLE1, the Star Server passes the request to the local DBMS, which gets the information from DBLOC_A1.

Example: Two Node Configuration

In the following diagram, the client and the Star Server are on the same node, Node A. One of the databases, DBLOC_A1, resides on Node A and the other database, DBLOC_B1, is on the remote Node B. The coordinator database resides, as always, on the same node as the Star Server:



The configuration requires Ingres Net to be installed on Node A and Node B. In addition, Ingres Star users on Node A must have access to an account on Node B and must have supplied Ingres Net with the account information using the Net utility netutil or must have entered the installation password to Node B on Node A with netutil. (For further information on netutil requirements in a distributed environment, see the *System Administrator Guide*.)

To invoke the Terminal Monitor on the distributed database, issue the following command at the operating system prompt:

```
sql distdb/star
```

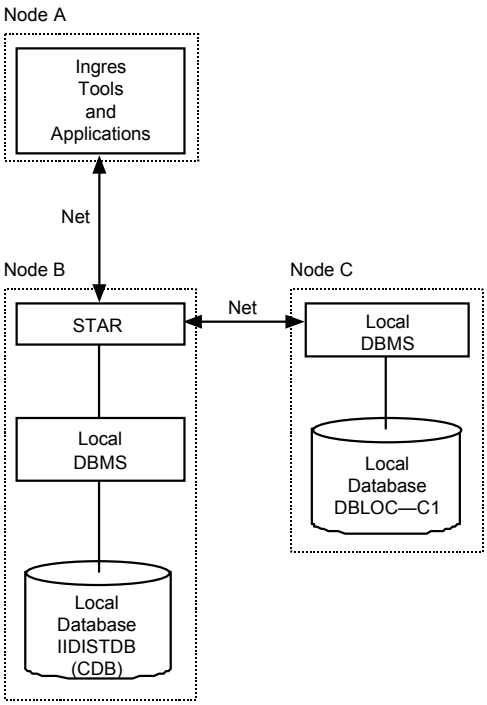
To incorporate TABLE2 in database DBLOC_B1 into the distributed database, issue the following statement in an application or an SQL session:

```
register table table2 as link
  with node = node_b,
  database = dbloc_b1;
```

When an Ingres Star user accesses TABLE2, Ingres Star requires a network connection to DBLOC_B1 to access TABLE2.

Example: Three Node Configuration

The following diagram shows the client on one node, Node A, the Star Server and the coordinator database on another node, Node B, and a local database, DBLOC_C1, on a third node, Node C:



This configuration requires Net on all three nodes and either:

- netutil entries for the user on Node A (authorizing a connection to Node B) and on Node B (authorizing a connection to Node C), or
- An installation password for Node B on Node A, and an installation password for node C on Node B

To invoke the Terminal Monitor on the distributed database, you must supply a node name as follows at the operating system prompt:

```
sql node_b::distdb/star
```

To incorporate TABLE3 in database DBLOC_C1 into the distributed database, issue the following statement in an application or an SQL session:

```
register table table3 as link
  with node = node_c,
  database = dbloc_c1;
```

When an Ingres Star user accesses TABLE3, Ingres Star requires a network connection to DBLOC_C1 to access TABLE3.

Distributed Transactions

A distributed transaction is a transaction that spans more than one local database. To the Ingres Star user, a distributed transaction looks like a transaction on a single local database. The user issues a single commit or rollback statement to end the distributed transaction. But a distributed transaction may actually involve multiple local transactions. If it does, Ingres Star coordinates these local transactions so that they behave like a single transaction. Therefore, either all of the data in a distributed transaction is committed or none of it is. Ingres Star uses a protocol known as a two-phase commit to ensure that this is what happens.

Two-Phase Commit

The term two-phase commit comes from the fact that there are two phases to committing a distributed transaction where two or more databases are updated. The two phases consist of:

- Agreement between all sites to commit
- Committing the updates

Star manages these two phases in the following way:

Phase 1

Phase 1 begins when the user issues a commit statement. Ingres Star sends a prepare-to-commit notice to each database involved in the distributed transaction. If all databases indicate that they are prepared to commit, Ingres Star makes the decision to commit the transaction. The local databases remain in the prepare-to-commit state and wait for Ingres Star's instruction to commit.

Phase 2

Ingres Star sends a commit to all sites involved in the transaction. Ingres Star guarantees that all sites will commit.

If the connection to a local database is lost between the time that Ingres Star decides to commit and the time the local database actually obeys that instruction, Ingres Star keeps trying to complete the transaction until the connection is restored and the commit is made. Ingres Star does not return control to the end user until all nodes have committed.

Notes:

- If any part of Phase 1 fails, for example, if Ingres Star loses a network connection to a node before all databases are prepared to commit, Ingres Star rolls back the transaction at all sites, including those that are already prepared to commit.
- If any part of Phase 2 fails, Ingres Star still eventually commits the transaction.

When Two-Phase Commit Is Not Used

Not all distributed transactions require two-phase commit. For example, a transaction that does not update, or that updates only one database, requires no coordination between databases. In this case, Ingres Star uses a single-phase commit that consists of sending commit messages to each database.

Sometimes a distributed transaction cannot use the two-phase commit protocol because one of the databases involved does not support it. For example, the following do not support two-phase commit:

- Some databases accessed using Enterprise Access products
- Ingres databases on Ingres cluster nodes

Such databases may still participate in a distributed transaction if their data is not updated, or if the databases are at the only site that is updated in a transaction. If only one site not capable of two-phase commit is involved in a multi-site update, Ingres Star will simulate two-phase commit using the protocol described in the Simulated Two-Phase Commit section.

Simulated Two-Phase Commit

In the case where an update is performed to a site that is not capable of two-phase commit and where a multi-site update transaction is required, Ingres Star simulates two-phase commit. It does this by first sending a prepare-to-commit to all sites that are capable of that protocol, then sending a commit to the single site not capable of two-phase commit, and finally sending commits to all other prepared sites. Note that Ingres Star only supports simulated two-phase commit when a single site not capable of two-phase commit is involved in a multi-site update. If more than one site not capable of two-phase commit is updated, Ingres Star refuses the attempted update.

StarView Utility

Ingres Star has a distributed database management utility called StarView, which lets the Ingres Star database administrator:

- Obtain information about the nodes, databases and tables that make up a distributed database
- Test the network connections between the nodes in the distributed database
- Register local tables in a distributed database and remove those registrations

For a thorough explanation of the StarView utility, see the chapter "Managing a Distributed Database with StarView."

Many Ingres Star operations can also be performed through the Visual DBA interface, which can be invoked from Ingres Visual Manager. For more information, see the online help system.

Query Optimizer

Ingres uses a query optimizer to develop sophisticated query execution strategies. The optimizer makes use of the following kinds of information:

- Table size
- Number of rows in a table
- Data-related information such as the amount of duplication in data values within a column and distribution of data values
- Network costing factor

Ingres Star maintains statistical data in a catalog named iistatistics, and data about the distribution of data in a table named iihistograms.

Ingres Star supports a distributed query optimizer to choose the most efficient access or query plan between different databases and between different nodes.

When developing the query plan, Ingres Star has the ability to move data rows from one site to another in order to maximize processing resources while minimizing data movement.

Ingres Star can analyze line cost factors between nodes and remote-node CPU processing performance and choose whether and where to transfer data for optimum performance.

For additional information on the Ingres query optimizer, see the *Database Administrator Guide*. The detailed steps for viewing query plans can be found in the online help.

Chapter 3: Preparing to Install Ingres Star

This chapter explains the various installation considerations that are needed while installing the Ingres Distribution Option. You have the option of installing Ingres Star during the standard Ingres installation procedure.

Installation Requirements

The following are requirements for installing Ingres Star:

- The Star Server must reside on the same node as the distributed database—the createdb *distdbname*/star command is executed on this node.
- The iidbdb (the Ingres master database) must be present before you install a Star Server.
  - If you are installing a new system, the iidbdb is created as part of the installation procedure.
  - For upgrading an installation, see the appendix "Release Compatibility."

VMS: In a cluster installation, a distributed database is accessible from all nodes in the installation but Net must be installed. For example, assume there are three nodes in the installation, Nodes A, B and C. If the distributed database is created on Node A, Ingres Star requires Net to access the distributed database from Nodes B and C.

Installation Prompts for Ingres Star

During installation you will be prompted for certain information needed by Ingres Star:

Net connections

Each Ingres Star user session requires one connection for each remote LDB being referenced while connected to Ingres Star. For example, if a user needs to connect to ten remote LDBs, you must establish ten outbound connections on the Ingres Star node, plus one inbound connection on each of the remote nodes.

If a distributed database has tables registered from 10 remote LDBs and all 10 remote LDBs are accessed by Ingres Star during the user's session, then 10 connections are required. If two users are to access this distributed database simultaneously, then 20 Net connections are required.

All connections are kept open until the user ends the session.

To specify or modify the number of inbound and outbound connections, see the *System Administrator Guide*.

Local Ingres DBMS connections

For each Ingres Star user session, three local sessions (connections) are required between the Star Server and the local DBMS for Ingres Star access to the coordinator database. For example, if the Star Server is configured for five user sessions, the local Ingres DBMS Server must be configured for at least 15 user sessions.

Star Server Startup

UNIX: The Star Server is a DBMS Server, which is started with the command:

```
ingstart -iistar
```

VMS: The Star Server is a DBMS Server, which started with the parameter:

```
ingstart /star
```

Operating System Requirements

The following are operating system requirements for Star:

UNIX: The executable setuid must be turned on. Only the user who owns the installation can start the Star Server. The install sets up the correct protections (that is, setuid) on the Star Server.

VMS: The Star Server requires the following privileges to run: PRMMBX, WORLD, READALL (for client/server communication), SYSPRV (to open error and status logging files), and SHARE (to open terminals for trace output). The vmsinstal utility installs the Star Server with all of the appropriate permissions.

How User Authorization to Nodes Is Established

Installation passwords can be used to set up authorizations for Ingres Star access, as discussed in Installation Passwords (see page 29).

You can make authorization entries in the netutil or ingnet utilities for Ingres Star access.

Installation Passwords

Installation passwords enable Net access between client and server without requiring each user to make a remote authorization. Additionally, installation passwords release Ingres Star clients from having to provide both local authorization (that is, authorization to the Ingres Star node) and remote authorizations to each remote node of a distributed database.

To use installation passwords, the system administrator gives a password to an Ingres server installation and enters the installation password on each trusted client. This then allows all users on the trusted clients to access the server installation without each having to provide a separate remote private authorization to the server.

How Installation Passwords Are Set Up

Installation passwords are set and entered with the netutil or ingnet utilities. To set a server's installation password, the system administrator uses the special username "*" for the server's local vnode. On each trusted client, the system administrator enters the installation password to the server's vnode, again using the special username "*". This enables client access to the server. There must also be a connection data entry at the trusted client pointing to the remote installation.

Here is an example of setting and entering an installation password. (This example uses netutil commands; alternatively, forms-based settings can be used.)

Nodes london and hongkong define hk_password as *'s installation password:

- On the hongkong node, set the installation password:

  ```
  create global login hongkong * hk_password
  ```

- On the london node, enter the installation password of hongkong:

  Note: This is the same command as on the hongkong node

  ```
  create global login hongkong * hk_password
  ```

- On the london node, create the connection data necessary for london's Communication Server to locate and connect to hongkong:

  ```
  create global connection hongkong hongkong tcp_ip fe0
  ```

In this command, the first hongkong specifies the vnode to which the connection is being made. The second hongkong specifies the node name or network address.

Users on london now have Net access to hongkong without having to define a private remote authorization.

For full details on using netutil for installation passwords, see the *System Administrator Guide*.

User Authorization Using Netutil and Ingnet

To access remote node connections, all Ingres Star users must run netutil or ingnet on the Ingres Star node and define their remote user authorizations to each remote node. To access a local Star Server, all Ingres Star users must run netutil or ingnet and define their logins to their accounts on their *local* node.

Users only need define their authorizations to each remote node and the local node once. The authorizations remain until the users choose to delete them with the netutil or ingnet utilities.

If a user's password is changed on one of the remote nodes, the corresponding netutil or ingnet authorization must be changed accordingly.

For complete details about the netutil or ingnet utilities, see the *System Administrator Guide* and online help.

Global and Private Authorizations

The authorizations defined in netutil or ingnet can be globally entered by the Net system administrator and used by all users, or they can be private entries.

Group Accounts

As an alternative to allowing each user private authorization to each remote node, the Net system administrator may define a login to serve as a group account for each local node and define global access to this account through netutil or ingnet. Ingres Star will connect to Ingres on the local node using the group account name and not the Ingres Star user's name.

For example, consider the following configuration:

- User dave and the Star Server reside on the london node. On node london, user dave has made no private authorization entries in netutil or ingnet other than to authorize his access to the london node.
- netutil or ingnet has a global authorization on london authorizing francis to access the paris node.

Dave uses Ingres Star on node london to access data from the paris node. Ingres Star's session with the database on paris will be in the name of francis; Dave is able to access only those tables with permissions granted to the francis login.

Private Accounts

In most cases, the individual user will want to define a private authorization to a node even if the node itself is defined globally. Defining authorizations for any node makes sense only if the authorized user has access to a login account on that node.

How User Authorization to the Local Node Is Established

The local vnode name is generated for you automatically when you install Ingres Star. The name is stored as the configuration parameter local_vnode. The local_vnode name is viewable through the CBF or Visual CBF utilities. For details, see the *System Administrator Guide.*

Authorization in Netutil and Inget for Recovery

For Ingres Star to perform two-phase commit recovery after a failure, there must be a netutil or ingnet authorization on the node where the Star Server resides for the owner of the Ingres installation or the Ingres Star installation. The netutil or ingnet authorization must be defined for every node referenced in each database accessed by the Star Server.

For example, Ingres Star running on node New York references nodes London, San Francisco, and Tokyo. On the New York node, there must be netutil or ingnet entries for the installation owner or the system administrator to all three remote nodes referenced (London, San Francisco, and Tokyo).

Authorization Examples

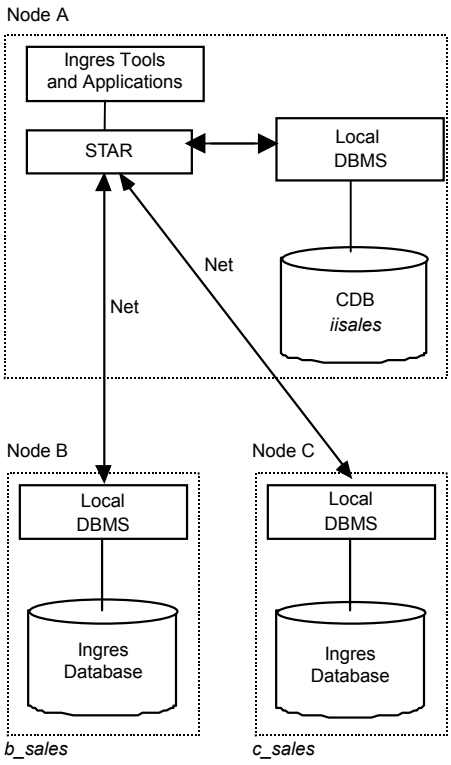
This section gives examples of using installation passwords or netutil or ingnet to define the network configuration of a Ingres Star database.

Example 1

In the following diagram, client and Ingres Star installation reside on node_A. Tables in the distributed database, node_A::sales/star, are registered from databases node_B::b_sales/ingres and Node_C::c_sales/ingres.

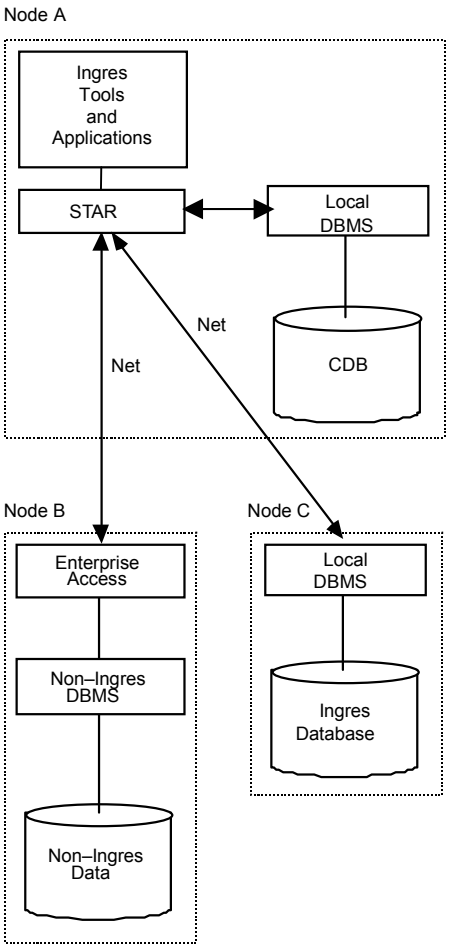
- To set installation passwords for node_B and node_C:
  - On node_B, set installation password to <b_password>.
  - On node_C, set installation password to <c_password>.
- To set installation passwords for node_B and node_C on node_A:
  - On node_A, enter <b_password> as installation password for node_B.
  - On node_A, enter <c_password> as installation password for node_C.

Users of node_A::sales/star can now access registered tables from node_B::b_sales/ingres and node_C::c_sales/ingres without each having to provide private remote authorization from node_A to both node_B and node_C.



Example 2

In the following diagram, the Ingres Star application and the Star Server both run on Node A. The Ingres Star distributed database consists of an SQL Enterprise Access on Node B and an Ingres local DBMS on Node C.



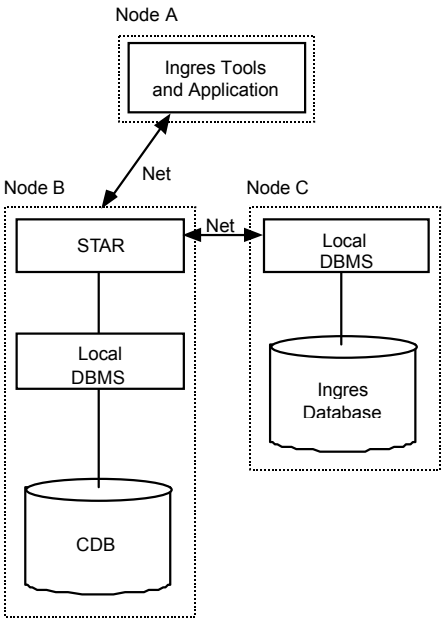
The netutil or ingnet requirements for running the application on Node A are as follows:

- On Node A:
  - There must be an authorization entry for a user on Node A. (Be sure that you do not overlook this authorization step.) The node name supplied to netutil or ingnet should be the same as the value of local_vnode, as described in Defining User Authorization to the Local Node.
  - There must be private or global entries for node definition and user authorization for Node B and Node C.
- On Node B and Node C:

  No netutil or ingnet definitions are required.

Example 3

In this example, the Ingres Star application runs on Node A and connects to a Star Server on the remote node, Node B. There is an Ingres local DBMS on Node C.



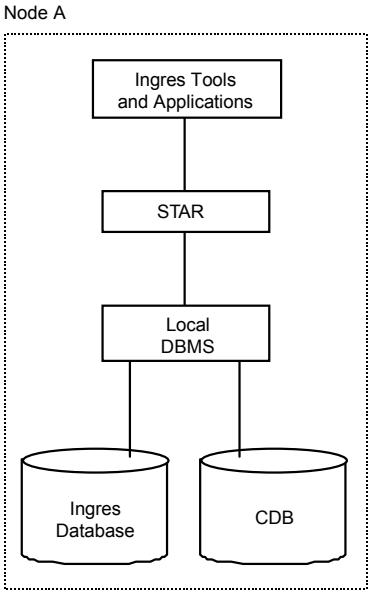
The netutil or ingnet requirements for running the application on Node A are as follows:

- On Node A:
  - There must be a node entry for Node B.
  - There must be an authorization entry for a user on Node B.
- On Node B:
  - There must be a node entry for Node C.
  - There must be an authorization entry for a user on Node C.

Note that netutil or ingnet must be used on both Node A and Node B.

Example 4

In this example, the Ingres Star application, the Star Server, and the local databases are all on the same installation on one node.



Chapter 4: Maintaining a Distributed Database

This chapter explains various aspects of maintaining a distributed database. It describes the statements and commands for creating, naming, and populating your distributed database.

Distributed Database Maintenance Tasks

A distributed database can be made up of tables, views, procedures, and indexes in local Ingres databases, remote Ingres databases, and non-Ingres databases accessible through Enterprise Access products.

Database maintenance tasks include:

- Creating distributed databases

  Note: You are not limited in the number of databases that you can create. You can create as many databases as your operating system allows.
- Destroying databases
- Creating tables and views on distributed databases
- Removing tables and views
- Registering (defining to Ingres Star) tables, views, and procedures (indexes are registered automatically)
- Refreshing registered tables or views
- Removing registrations

VDBA and Distributed Database Maintenance

You can use Visual DBA to maintain a distributed database. Use the Databases branch in the Database Object Manager window.

The following topics in the VDBA online help contain procedures to maintain distributed databases:

- Creating a Database

  Note: To create a database, you need the createdb privilege. This subject privilege is granted by default to the system administrator, who in turn can grant it to other users, such as database administrators, who need to create databases. For more information on subject privileges, see the *Database Administrator Guide*.
- Dropping a Database
- Creating a Table on a Distributed Database (choose Creating a Table first)
- Creating a View
- Dropping a Table
- Dropping a View
- Registering a Distributed Table or View
- Registering a Distributed Procedure
- Refreshing Registered Tables and Views
- Removing Registrations

Commands for Performing Distributed Database Maintenance

Distributed databases can also be created and destroyed by the createdb and destroydb commands, which are executed at the operating system level. For more information, see Createdb Command (see page 44) and Destroydb Command (see page 45).

Statements for Performing Distributed Database Maintenance

Some tasks can be accomplished using statements, which can be issued from an interactive session with SQL or ISQL, or from an application.

These statements are as follows:

- The create statement to create new tables and views in the distributed database and automatically register them
- The drop statement to remove tables and views from the distributed database
- The register as link statement to define existing tables, views, indexes, and database procedures as component objects of the distributed database
- The register as link with refresh statement to register local updates to tables, views, and indexes that are part of the distributed database
- The remove statement to remove the registration of database objects previously incorporated with the register and create statements

By default, the data definition language (DDL) statements create, drop, register, and remove are all committed immediately, independently of the user's transaction. To learn how to turn off concurrency, see DDL Concurrency Mode (see page 70).

Naming Conventions

For details on syntax rules regarding names, see object names in the *SQL Reference Guide.*

Database Naming Restrictions

Database names must be unique to 32 characters (or the maximum file length imposed by your operating system, if less than 32). In addition, the name of your distributed database must be unique at the coordinator node; a distributed database and a local database cannot coexist using identical names.

Check the Enterprise Access documentation specific to your Enterprise Access for any naming restrictions that apply to the non-Ingres database your distributed database is accessing.

Sizing Attributes

Ingres Star allows you flexibility in certain sizing extensions.

If Ingres Star addresses a local DBMS that supports increased sizes of rows and the varchar data type, it supports these sizes up to a limit of 4096 bytes.

Server Class

Many of the commands used in creating your database require you to specify the type of local DBMS that contains the tables or views you are registering or creating:

- The *server_class* you specify on the createdb command must be star.
- On the statements for registering and creating tables and views, Ingres *server_classes* as well as Enterprise Access *server_classes* may apply.

For the name of the Enterprise Access server class, see the *Enterprise Access Installation and Operations Guide*.

Case

Names that identify database objects may be uppercase, lowercase, or mixed case. In an Ingres Star database, the case of the object's name as stored in the Ingres Star catalogs depends on:

- The case-translation semantics established when the database was created
- How the user specifies the name of the object when it is created
- Whether or not the user delimits the name of the object

Mixed-case names are allowed if the distributed database has been created specifying mixed-case delimited identifiers. For examples, see Naming Registration Examples (see page 41). In addition, an object name must be delimited to preserve case.

Mixed-case names are not allowed if delimited identifiers are not used when registering an object. The case of the name will be uppercase or lowercase, depending on the case of regular identifiers specified at installation time.

In general, Ingres Star performs case translation in order to map the name of a registered object into the name of a local object. However, there are some restrictions on incorporating objects from mixed-case local databases or Enterprise Access products into a Ingres Star database that has been created without mixed-case delimited identifiers. For more information on delimited identifiers and Ingres Star, see Installation Options (see page 41).

Installation Options

Ingres Star object names may be case-significant, depending on an option specified during installation. The option specifies the rules that the local Ingres DBMS follows in translating identifiers supplied by the user. (An identifier is the syntactic component of an SQL statement that represents the name of a database object.) There are two types of identifiers:

- A *regular identifier* is a name or word.
- A *delimited identifier* is a word or words delimited by double quotes. It allows extended characters and mixed case.

Option Name	Regular Identifiers	Delimited Identifiers
Lowercase option (Ingres setting)	Convert to lowercase	Convert to lowercase
ISO Entry SQL92 standard	Convert to uppercase	Retain case, even if mixed

With the lowercase option (Ingres setting), createdb creates local and distributed databases whose translation rule is to put into lowercase both regular and delimited identifiers. To create an object with a mixed-case name, you must use the ISO Entry SQL92 standard option, and you must use a delimited identifier to specify the object name.

For a further description of identifiers, see the *SQL Reference Guide*.

Naming Registration Examples

An Ingres Star database may consist of objects registered from remote databases whose case-translation rules differ from those of the distributed database. These examples illustrate those situations. (These naming examples use the register as link statement. For details on this statement, see Register as Link Statement (see page 46).

Example: Default DDB and ISO-Compliant LDB

Assume that the distributed database was created with the lowercase option (Ingres setting), which puts both regular and delimited identifiers in lowercase. Assume that the local database is ISO Entry SQL92-compliant. The local database, iso_db, was created with case-translation rules of uppercase regular identifiers and mixed-case delimited identifiers.

In this example iso_db contains a table named PARTS, and the user wishes to register that table into the distributed database. The following statement creates a distributed object named parts. It will correctly map all queries against parts to the table PARTS in the iso_db database:

```
register parts as link
  with node=node_A, database=iso_db;
```

Assume that table PARTS has columns named PART_NO and PRICE. At registration time, the Star Server translates the names of these columns before adding them to the Ingres Star catalog iicolumns as columns part_no and price. In addition, the following user's query against the distributed database will correctly map price and part_no to uppercase names before querying the PARTS table:

```
select price from parts where part_no=40754;
```

Example: Mixed-Case Names

Ingres Star employs the mapping rules described in Example: Default DDB and ISO-Compliant LDB (see page 42) if the local database names are mixed case. There is, however, a restriction on registering local database tables with mixed-case names into a distributed database that does not support mixed-case names. Assume that you wish to register the Low Budgets table from the iso_db database into your default distributed database. Mixed-case names may be specified only using a delimited identifier. But the following statement produces an error:

```
register low_budgets as link from "Low Budgets"
  with node=node_A, database=iso_db;
```

Because the distributed database translates all identifiers to lowercase, the Star Server attempts to look up the table low budgets in the catalogs of the iso_db database. It does not find that table because the mixed-case property of the table's name has already been lost. Therefore, Ingres Star rejects any register statement that delimits the name of the local database object or owner if the distributed database does not support mixed-case delimited identifiers but the local database does.

As a workaround to this restriction, the from clause on the register statement allows you to specify the local owner name and the local table name as a string constant. The Star Server does not apply case-translation rules to string constants. Thus the following statement would succeed in registering the table Low Budgets from a case-sensitive local database into a case-insensitive distributed database:

```
register low_budgets as link from “Low Budgets”
  with node=node_A, database=iso_db;
```

Example: DDB with Mixed-Case Delimited Identifiers

There is no restriction if the distributed database has been created to support mixed-case delimited identifiers, whether or not the local database supports them.

Assume that you have a table named corp_dept in a local database lower_db that supports only lowercase names. Assume that your distributed database supports mixed-case names and that you wish to register corp_dept as Corp_Dept in the distributed database. You issue the statement:

```
register "Corp_Dept" as link
  with node=node_A, database=lower_db;
```

Ingres Star maps the distributed database name Corp_Dept into corp_dept when it queries the lower_db database. If corp_dept has columns dno, name and budget, Ingres Star records these in the distributed database's iicolumns catalog using the case rules of regular identifiers. If the distributed database was created specifying uppercase translation for regular identifiers, the column names of Corp_Dept will appear as DNO, NAME and BUDGET in the distributed database.

If you wish the registration to have mixed-case column names, you can specify them in the register statement:

```
register "Corp_Dept" ("Dno", "Name", "Budget") as link
  with node=node_A, database=lower_db;
```

Createdb Command

You can use the createdb command to create a distributed database as you would a local database, except that you must add /star to the distributed database name.

The createdb command does the following:

- Creates an Ingres database and identifies it as the coordinator database (see Ingres Star Components (see page 15).
- Builds and populates Ingres Star-specific catalogs in the coordinator database just created
- Generates iidbdb entries to describe the distributed database and the coordinator database and to associate the two

For flags and full syntax details of the createdb command, see the *Command Reference Guide.*

Examples: Createdb

The following command creates a distributed database named corporateddb:

```
createdb corporateddb/star
```

Because a coordinator database name was not specified, the coordinator database name defaults to iicorporateddb. When createdb finishes, there will be a single new database (the coordinator database iicorporateddb) in the Ingres data location. There will be entries in the iidbdb for the distributed database and the coordinator database.

Note: There will not be a physical database for the distributed database, but there will be iidbdb catalog entries associating the coordinator database with the distributed database.

The following command creates a distributed database named corporateddb with a coordinator database named corporate that overrides the automatically generated coordinator database named iicorporateddb:

```
createdb corporateddb/star corporate
```

The following command creates a distributed database named corporateddb with a coordinator database named corporate. It assigns the default data location to corp_loc and installs the Ingres/Vision client catalogs so that users can access Vision on the corporateddb distributed database.

```
createdb -dcorp_loc corporateddb/star corporate -f vision
```

Destroydb Command

You can use the destroydb command to delete a distributed database, as you would for a local database.

The destroydb command removes the distributed database, the coordinator database, and all the Ingres Star objects that make up the distributed database. Data in underlying tables in non-coordinator local databases registered in the distributed database are not affected.

For more information on the destroydb command see the *Command Reference Guide.*

Example: Destroydb

This command removes a distributed database named corporateddb:

```
destroydb corporateddb
```

Register as Link Statement—Define Database Objects to Ingres Star

The register as link statement defines a distributed database component name to Ingres Star as an alias of a local or remote table or view. The statement is stored in the iiregistrations catalog.

Use the register as link statement to define to Ingres Star:

- Data resident in an Ingres local database
- Database procedures resident in an Ingres local database or accessible through Enterprise Access
- Data accessed by means of Enterprise Access

Note: To register non-SQL data, you first must identify the data to the non-SQL Enterprise Access with the register as import statement. For information on how to use this statement, see the guide for your specific Enterprise Access.

The register as link statement has the following format:

```
register object_type object_name
       [(col_name {, col_name})]
       as link
       [from [local_owner_name.]local_object_name]
              [with
                     [node = node_name,
                     database = database_name]
                     [, dbms = server_class]]
```

object_type

Can be table, view or database procedure. For tables or views, the object_type is optional. However, to register a database procedure, the keyword procedure must be specified.

object_name

Is the Ingres Star name of the local DBMS table, view, or database procedure you are registering. It may be delimited with double quotes.

col_name

Is the Ingres Star name of the corresponding column in the local DBMS table or view. (Do not use when registering procedures.) It may be delimited with double quotes.

There can be as many column names as Ingres Star allows in a table or view. This column name is the one used in queries presented to Ingres Star. All column names must follow the Ingres naming conventions.

This column name must be specified in the order corresponding to its location in the underlying local DBMS table or view (the nth Ingres Star column name corresponds to the nth local column name.)

All columns must be named. If column names are not specified, Ingres Star uses the column names obtained from the local DBMS standard catalogs for the table or view specified.

as link

Indicates that the register statement will create an Ingres Star registration linked to a table, view, or database procedure in a local database. This clause must be specified.

from

Specifies the owner and name of the local table, view, or database procedure you are registering. You must use this clause if you are registering an object in your distributed database with a different name from its name in the local database, or the object name does not follow Ingres naming conventions, is case sensitive, or is owned by another user.

local_object_name

Is the name of the table, view, or database procedure in the local database.

It may be delimited with double quotes to preserve case and allow special characters. However, if the Ingres Star database does not support mixed-case delimited identifiers and the local DBMS or Enterprise Access product does support mixed-case delimited identifiers, you should use a single-quote delimiter in order for Ingres Star to preserve the case.

local_owner_name

Is the name of the owner of the object in the local database. You can register other user's tables, views, and database procedures, but you must specify the owner name. If you do not specify an owner name, local_owner_name defaults first to an object owned by the current user, second to an object owned by the DBA, and third to an object owned by $ingres.

It may be delimited with double quotes to preserve case and allow special characters. However, if the Ingres Star database does not support mixed-case delimited identifiers and the local DBMS or Enterprise Access product does support mixed-case delimited identifiers, you should use a single-quote delimiter for Ingres Star to preserve the case.

Do not forget to include the period at the end of the owner name.

with

Provides additional information about the local database table, view, or database procedure being registered and its location in relation to Ingres Star.

node = *node_name*

Is the Net vnode name (the virtual node name) that holds the object you are registering. The default is the node that contains the Star Server processing the command. The node_name may be up to 32 characters long. It may be delimited with double quotes.

If you specify the node = clause, you also must specify the database = clause.

database = *database_name*

Specifies the name of the local database that contains the object you are registering. It may be delimited with double quotes.

The database name may be up to 256 bytes long provided it is specified in quotes.

If you specify the database = clause, you must specify the node = clause. If the database is omitted, the coordinator database is the default.

dbms = *server_class*

Specifies the type of local DBMS that contains the object being registered.

The server_class must be Ingres or one of the SQL Enterprise Access products. If you do not specify a server class the default is the value in default_server_class on the remote installation (Ingres, unless defined otherwise.) Use the Configure Name Server screen of the CBF or Visual CBF utilities to view or change this value.

Register Table as Link Statement—Define Table to Ingres Star

The register table as link statement defines an already existing local DBMS table to Ingres Star. No new table is created. All secondary indexes associated with the table are registered automatically when you register a table in your distributed database if the table is in a local database.

The register table as link statement has the following format:

```
register [table] table_name
       [(col_name {, col_name})]
       as link
       [from [local_owner_name.]local_table_name]
       [with
              [node = node_name,
              database = database_name]
              [, dbms = server_class]]
```

table

Is an optional object type identifier. Whether or not you specify table, Ingres Star queries the local DBMS and determines the object type.

If you specify table, Ingres Star issues an error if the local DBMS type is not a table.

table_name

Is the Ingres Star name of the local DBMS table you are registering.

It may be delimited with double quotes to preserve case and allow special characters. However, if the Ingres Star database does not support mixed-case delimited identifiers and the local DBMS or Enterprise Access product does support mixed-case delimited identifiers, you should use a single-quote delimiter in order for Ingres Star to preserve the case.

This name must follow Ingres naming conventions. It will appear in the Ingres Star catalogs as a table. In the table_subtype column of the iitables standard system catalog, it will have a subtype of L (registered as Link). For information, see the chapter "Understanding Ingres Star Catalogs."

local_table_name

Is the name of the table in the local DBMS.

It may be delimited with double quotes to preserve case and allow special characters. However, if the Ingres Star database does not support mixed-case delimited identifiers and the local DBMS or Enterprise Access product does support mixed-case delimited identifiers, you should use a single-quote delimiter in order for Ingres Star to preserve the case.

The default is to use the distributed database table_name you specify as the name of the local DBMS table that you are registering.

Enter this name if you are registering the table in your distributed database with a name different from its name in the local DBMS or if the table name in the local DBMS does not follow naming conventions and/or is case sensitive or is owned by another user.

Note: This table name must be the actual base table name, not a synonym.

The elements that are common to all three register as link statements are described in Register as Link Statement (see page 46).

Examples: Register Table as Link

Consider the following database configurations, used in examples in this and following sections:

- The distributed database corporateddb resides on the london node with its coordinator database corporate. The table, prospects, has been created locally in corporate.
- A remote database pacific is on a node tokyo and contains a table, customers.
- A second remote database west_usa resides on the node reno and contains a table, sales and a view, follow_ups. The table, sales belongs to user john and has columns customer, invoice_number and total.

The following statement issued in a session with the distributed database corporateddb registers the prospects table from the coordinator database in corporateddb:

```
register prospects as link;

This statement registers prospects in corporateddb under the name west_prospects:

register west_prospects as link
  from prospects;
```

This statement registers the table, sales from the database west_usa on the nod, reno, giving it the name usa_sales and referring to its columns as customer, inv_no and amount:

```
register table usa_sales
  (customer, inv_no, amount) as link
  from john.sales
  with node = reno, database = west_usa;
```

If the distributed database corporateddb is case sensitive and you wish to create registrations with case-sensitive names, or if you wish to include special characters in the registration name, you would use a delimited identifier to specify the registration. The following example shows how you would register a table named usa sales into corporateddb:

```
register table "usa sales"
  (customer, "inv no", amount) as link
  from john.sales
  with node = reno, database = west_usa;
```

If the west_usa database allows mixed-case identifiers, or if the local table name includes mixed-case or special characters, you would use a delimited identifier in the from clause. For example, john's table may be named Sales.

```
register table usa_sales
  (customer, inv_no, amount) as link
  from john."Sales"
  with node = reno, database = west_usa;
```

Note: You can use a delimited identifier on the from clause only if the case-translation semantics of both the distributed database and the local database are compatible. Ingres Star rejects the above statement if corporateddb does not support mixed-case identifiers and west_usa does support them. The reason for this is that the identifier Sales will be case converted by the Star Server, and any mixed-case characters will have been converted before the registration is processed. In this instance, you would use single quotes. The Star Server does not translate singly-quoted strings. For example:

```
register table usa_sales
  (customer, inv_no, amount) as link
  from john."Sales"
  with node = reno, database = west_usa;
```

Register View as Link Statement—Define View to Ingres Star

The register view as link statement defines an already existing local DBMS view to Ingres Star. No new view is created.

The register view as link statement has the following format:

```
register [view] view_name
         [(col_name {, col_name})]
         as link
         [from [local_owner_name.]local_view_name]
         [with
                  [node =node_name,
                  database =database_name]
                  [, dbms = server_class]]
```

view

An optional object identifier. Whether or not you specify view, Ingres Star queries the local DBMS and determines the object type.

If you specify view, Ingres Star issues an error if the local DBMS type is not a view.

view_name

The Ingres Star name of the local DBMS view you are registering.

It may be delimited with double quotes to preserve case and allow special characters. However, if the Ingres Star database does not support mixed-case delimited identifiers and the local DBMS or Enterprise Access product does support mixed-case delimited identifiers, you should use a single-quote delimiter in order for Ingres Star to preserve the case.

This name must follow Ingres view naming conventions. It will appear in the Ingres Star catalogs as a view. In the table_subtype column of the iitables standard system catalog, the Ingres Star view name will have a table_subtype of L (registered as Link). (For more information on catalogs, see the chapter "Understanding Ingres Star Catalogs.")

local_view_ name

The name of the view in the local DBMS.

It may be delimited with double quotes to preserve case and allow special characters. However, if the Ingres Star database does not support mixed-case delimited identifiers and the local DBMS or Enterprise Access product does support mixed-case delimited identifiers, you should use a single-quote delimiter in order for Ingres Star to preserve the case.

The default is to use the distributed database view_name you specify as the name of the local DBMS view that you are registering.

Enter this name if you are registering the view in your distributed database with a name different from its name in the local DBMS or if the view name in the local DBMS does not follow Ingres naming conventions and/or is case sensitive or is owned by another user.

Note: This name must be the actual base name, not a synonym.

The elements that are common to all three register as link statements are described in Register as Link Statement (see page 46).

Example: Register View as Link

This statement registers the view follow_ups from the database west_usa on the node reno, and gives it the name usa_visits:

```
register view usa_visits as link
  from follow_ups
  with node = reno, database = west_usa;
```

Register Procedure as Link Statement—Define Procedure to Ingres Star

The register procedure as link statement defines an already existing local DBMS database procedure to Ingres Star. No new database procedure is created.

The register procedure as link statement has the following format:

```
register procedure procedure_name
         as link
         [from [local_owner_name.]local_procedure_name]
         [with
                  [node = node_name,
                  database = database_name]
                  [, dbms = server_class]]
```

procedure

Identifies the object type

procedure_name

The Ingres Star name of the local DBMS database procedure you are registering.

It may be delimited with double quotes to preserve case and allow special characters. However, if the Ingres Star database does not support mixed-case delimited identifiers and the local DBMS or Enterprise Access product does support mixed-case delimited identifiers, you should use a single-quote delimiter in order for Ingres Star to preserve the case.

local_procedure_name

The name of the database procedure in the local DBMS.

It may be delimited with double quotes to preserve case and allow special characters. However, if the Ingres Star database does not support mixed-case delimited identifiers and the local DBMS or Enterprise Access product does support mixed-case delimited identifiers, you should use a single-quote delimiter in order for Ingres Star to preserve the case.

The default is to use the distributed database *procedure_nam*e you specify as the name of the local DBMS database procedure that you are registering.

Enter this name if you are registering the database procedure in your distributed database with a name different from its name in the local DBMS or if the name in the local DBMS does not follow naming conventions, is case sensitive, or is owned by another user.

The elements that are common to all three register as link statements are described in Register as Link Statement (see page 46).

Examples: Register Procedure as Link

This statement registers LDB1 procedure p2 (on node node_A) using the same name in Ingres Star:

```
register procedure p2 as link
  with node=node_A, database=LDB1;
```

This statement registers LDB1 procedure p3 using the name proc2 in Ingres Star:

```
register procedure proc2 as link from p3
  with node=node_a, database=LDB1;
```

Catalogs for the Register Statement

The text for the register as link statement is stored in the iiregistrations standard catalog.

Execute Immediate Statement--Execute Register as Link Statement Dynamically

You can execute the register statement through dynamic SQL by the execute immediate statement.

For example, the following command registers the table customers from the database pacific on the node tokyo:

```
exec sql execute immediate
  'register customers as link
  with node = tokyo, database = pacific';
```

Note: Do not use the prepare statement for executing the register as link statement. Use execute immediate as described here.

Register as Link with Refresh Statement—Refresh Registration

Schema information in local tables can change often. Such information may relate to storage structure, unique keys, secondary indexes, number of columns, and statistics.

When schema or related table information changes in a local database that is part of a distributed database, the Ingres Star catalogs must be updated. Except for row and page counts, this is not done automatically. It must be done with the register as link with refresh statement.

For Ingres Star to take advantage of performance tuning at the local database level, tables affected should be refreshed using the register as link with refresh statement. This allows Ingres Star to use the latest table structure and secondary indexes when determining distributed query execution plans.

The register as link with refresh statement has the following format:

```
register object_name as link with refresh
```

object_name

Is the registered view or table name whose registration you are refreshing. It may be delimited with double quotes.

Register as Link with Refresh Restrictions

Register as link with refresh refreshes all the information about a table or view in the Ingres Star catalogs with information from the local database.

Views built on top of the table or view are unaffected. The table or view retains the same table_id or view_id.

The register with refresh statement is not allowed in the following circumstances (an error message is generated):

- If the local database table has fewer columns than when initially registered
- If the local database table columns have different data types from when initially registered
- If column name mapping was used when the table was initially registered and the local database table now has a different number of columns from when it was initially registered
- If either the distributed database or the local database (but not both) supports mixed-case delimited identifiers, Ingres Star internally maps column names. In this case, register with refresh is not allowed if the local database table now has a different number of columns from when it was initially registered

Register with refresh may be used on objects registered by the register statement or implicitly registered by Ingres Star. A table or view refreshed in this way retains the same table_subtype value in the iitables catalog, that is, it remains registered native if originally registered native. (A *native* object means the object was created through Ingres Star rather than merely registered through Ingres Star.)

Effects of Register as Link with Refresh

Several situations may arise that will require you to refresh the registration of a local DBMS object.

Where the old schema of the local table is identical to, or is a subset of, the new schema (for example, when a new column is added to the old local table and the table's creation time remains unchanged), register as link with refresh has the following effects:

- All views dependent on the Ingres Star-level table are retained.
- The statistics associated with the registration are left intact if the local table has no statistics or its statistics cannot be read because of the local database's different architecture. Otherwise, statistics from the local database replace existing Ingres Star-level statistics.

Where the local table has been dropped and recreated (shows a new creation time) using the same schema, possibly with new columns at the end, register as link with refresh has the following effect:

- All the views dependent on the Ingres Star-level table are retained.

Where the old schema of the local table is different from the new schema, for example, a column has been dropped or a column type has been redefined, then register as link with refresh is not allowed. An error is returned to the user.

Example: Register as Link with Refresh

The following example shows the use of register as link with refresh:

```
register west_prospects as link
  from prospects;

create view leads as
  select * from west_prospects;
commit;

/* Initiate connection to coordinator database */
direct connect;
drop prospects;

/* recreate with the same schema but with an additional column */
create table prospects;

/* return control to Ingres Star */
direct disconnect;

register west_prospects as link with refresh;

/* view created above */
select * from leads;
```

The result set from the view selection will show the original schema of the prospects table. The original target from the view is used for selection, and that target list contains only information from the old schema. However, a select on the registered table west_prospects results in a set of rows under the new schema.

For detailed descriptions of the direct connect/disconnect statements, see Connecting Directly to a Local Database (see page 72).

Using Register with Enterprise Access Products

To register an SQL Enterprise Access table or view in a distributed database, use the register as link statement.

Registering a non-SQL Enterprise Access object in a distributed database is a two step process:

- At the Enterprise Access, you first must use the register as import statement to register the Enterprise Access object in the non-SQL Enterprise Access. The Enterprise Access object then has an Ingres object name and looks exactly like an Ingres object. Then, exit from the Enterprise Access.
- Accessing the Ingres Star distributed database, you then use the register as link statement to register those Ingres names in your distributed database.

For a full explanation of the register as import statement, see the guide for your specific non-SQL Enterprise Access.

Remove Statement—Remove Registration

The remove statement removes registrations of tables, views, and database procedures from your distributed database. An object that is removed by the remove statement remains intact in the underlying local database, but is no longer identified to Ingres Star.

Local objects are registered by you with the register as link statement. You use the remove statement to remove these registrations. Only the registration is deleted. The local objects are not affected.

When you use the create table statement at the Ingres Star level, the table is automatically registered in your distributed database. You can use remove to delete these automatic registrations also. The underlying table remains.

You can execute the remove statement through dynamic SQL by the prepare/execute, and execute immediate statements.

The remove statement deletes only the registration residing in the Ingres Star catalogs. The table or view in the local DBMS is not affected.

You cannot remove an index with the remove statement.

You cannot remove a distributed view with the remove statement. You must use the drop statement to delete a distributed view.

The remove statement has the following forms:

- remove *object_name*
- remove table *table_name*
- remove view *view_name*
- remove procedure *procedure_name*

If the type of object is not specified, Ingres Star assumes it is a table or view. The keyword procedure must be given to remove a procedure name.

Remove Table Statement

The remove table statement removes from the Ingres Star catalogs table definitions that were manually registered using register table as link, or which were automatically registered using create table at the Ingres Star level.

The local DBMS table itself is not changed, but all the Ingres Star catalog definitions are deleted, including any related view or index definitions.

The remove table statement has the following format:

```
remove [table] table_name
```

table

Is an optional object identifier. If you do not specify table, the registration of table_name is deleted whether the object is a table or a view. If table is specified, the object named must be a table.

table_name

Is the registered Ingres Star name of the table. It may be delimited with double quotes.

Example: Remove Table

To remove the registration of a table named west_prospects from your distributed database, use the statement:

```
remove table west_prospects;
```

To remove the registration named usa sales, delimit the table name. Use delimited identifiers only if you used them on the corresponding register statement:

```
remove table "usa sales";
```

Remove View Statement

The remove view statement removes from the Ingres Star catalogs view definitions that were registered using register view as link.

The remove view statement has the following format:

```
remove [view] view_name
```

view

Is an optional object identifier. Whether or not you specify view, Ingres Star queries the local DBMS and determines the object type.

If you do not specify view, view_name is deleted whether the object is a table or a view. If view is specified, the object named must be a view.

view_name

Is the registered Ingres Star name of the view. It may be delimited with double quotes.

Example: Remove View

To remove the registration of a view named usa_visits from your distributed database, use the statement:

```
remove view usa_visits;
```

Remove Procedure Statement

The remove procedure statement removes from the Ingres Star catalogs procedure definitions that were manually registered using register procedure as link.

The local DBMS procedure itself is not changed, but all the Ingres Star catalog definitions are deleted.

The remove procedure statement has the following format:

```
remove procedure procedure_name
```

procedure

Identifies the object as a database procedure

procedure_name

Is the registered Ingres Star name of the database procedure. It may be delimited with double quotes.

Example: Remove Procedure

To remove the registration of a database procedure named west_prospects_proc from your distributed database, use the statement:

```
remove procedure west_prospects_proc;
```

Create Statement

You use the create statement with Ingres Star to create new tables and views and add them to your distributed database.

- The create table statement creates a local table and registers the table in your distributed database as native.

  If you do not use the create table statement at the Ingres Star level, you first must create a table locally and then register it in your distributed database.

- The create view statement creates a distributed view.

Create Table Statement

Use the create table statement to create new tables:

- create table with *with_clause*
- create table as *subselect* with *with_clause*

The create table statement creates an object of type table. The table is stored in a local database or the coordinator database, depending on the *node_name* specified in the with clause. By default, the table is stored in the coordinator database. The table is automatically registered in the Ingres Star catalogs as a native object. Tables registered as native are distinguished from tables registered as links by the value in the table_subtype column in the iitables catalog.

Create Table With Syntax

The create table with statement has the following format:

```
create table table_name
       (col_name format {, col_name format})
       [with
              [node =node_name,
              database = database_name]
              [, dbms = server_class]
              [, table =local_table_name]
              [, LDB with clauses]]
```

Create Table as Subselect With Syntax

The syntax for the create table as *subselect with* statement is:

```
create table table_name
       [(col_name {, col_name})]
       as subselect
       [with
              [node =node_name,
              database =database_name]
              [, dbms = server_class]
              [, table = local_table_name]
              [, LDB with clauses]]
```

Create Table Syntax Elements

The *col_name*, *node_name*, *database_name*, *server_class*, and *local_table_name* syntax elements are as described in previous statements. The remaining syntax elements are described in the table below and in the following section, LDB With Clauses.

table

The object identifier. This must be specified.

table_name

The name in the distributed database. It may be delimited with double quotes.

If the local_table_name is not specified the registered table_name is used. The table names referenced in the subselect clause are Ingres Star-level objects. They must be registered or created first through Ingres Star before being called by subselect

format

Formats refer to the data type of the column as well as how unspecified values (blanks and nulls) should be handled. For a full description of data formats, see the *SQL Reference Guide*.

LDB With Clauses

Ingres Star recognizes and processes with clauses for defining *node_name*, *database_name*, *server_class*, and *local_table_name* as shown in the syntax descriptions above.

Other with clauses (*LDB with clauses*) in the syntax descriptions above) are not supported or recognized by Ingres Star. For example, Ingres Star is not responsible for handling location, journaling and duplicates in the with clause. However, it is Ingres Star's responsibility to guarantee that these clauses are properly transmitted to the local DBMS for processing. These options are received and managed by each local DBMS.

For a complete list of the *LDB* with *clauses*, see the with clauses of the create table description in your query language reference guide.

Examples: Create Table

The following statement creates the table corp_dept in the coordinator database and registers it in the distributed database:

```
create table corp_dept
    (dno char(8),
    name char(10),
    budget integer);
```

Create the table department in the database west_usa on the node reno and register it in the distributed database corporateddb under the name corp_dept:

```
create table corp_dept
    (dno char(8),
    name char(10),
    budget integer)
  with node = reno,
  database = west_usa,
  table = department;
```

A table low_budgets could then be created in the coordinator database and registered in the distributed database by using a subselect from the table corp_dept:

```
create table low_budgets
  as select * from corp_dept
  where budget < 10000;
```

The following example shows a create table statement with an *LDB* with *clause*, journaling:

```
create table corpemployee
    (name char(20),
    sal money)
  with journaling;
```

Journaling is set at the local DBMS. Ingres Star does not register this attribute in the Ingres Star catalogs but passes it to the local DBMS to be processed.

If your Star database allows mixed-case delimited identifiers or if you wish your table or column names to include special characters such as spaces, use a delimited identifier:

```
create table "World Wide Sales" ("Region" char(20),
  "Gross Sales in Millions" decimal(16,2));
```

Create View Statement

The create view statement creates a distributed view.

The created view is not a local database view. It is a distributed view known only to the distributed database.

Create View Syntax

The create view statement has the following format:

```
create view view_name
       [(col_name {,col_name})] as subselect
```

view

Refers to the object identifier. This must be specified.

view_name

Defines the name of the view in the distributed database. It can be delimited with double quotes. This is an Ingres Star-level object as opposed to an object in a local database. The definition of this view is entered into the Ingres Star catalogs. The view can reference other tables and views.

For a description of the *subselect* and other syntax details of the create view statement, see the *SQL Reference Guide*.

Drop Statement

Use the drop statement to remove tables and views from your distributed and local databases. You may only drop an object (table or view) that was created in Ingres Star.

Drop Table Statement

The drop table statement has the following format:

```
drop [table] table_name
```

Use this statement on table objects created by the create table statement at the Ingres Star level. It removes the actual table, its registration, and all its data, as well as the description of the table in the Ingres Star catalogs.

Important! Use the drop statement with care because this statement destroys objects. All dependent objects, for example, views and indexes, are also dropped.

Example: Drop Table

To remove the registration of corp_dept from the distributed database and the table itself from the local database, west_usa, use the following command:

```
drop table corp_dept;
```

To drop the table named World Wide Sales, delimit the table name:

```
drop table "World Wide Sales";
```

Drop View Statement

The drop view statement has the following format:

```
drop [table] view_name
```

Use this statement to remove a distributed view created by the create view statement at the Ingres Star level. It removes the description of the view from the Ingres Star catalog. It does not affect the underlying data on which the view is defined.

Table Registration Using StarView

As an alternative to registering existing tables and views from the command line, the StarView utility provides a forms-based interface to carry out this task.

Simply run StarView to display the tables and views in your existing databases, and choose the ones you wish to register in your distributed database.

DDL Concurrency Mode

The data definition language (DDL) statements register, remove, create, and drop cause updates to occur to the Ingres Star catalogs.

In order to increase multi-user concurrency and reduce the chance of deadlocks, these DDL statements are all committed immediately, independently of the user's transaction. Even if a user's transaction is subsequently aborted, the DDL statement is not aborted.

This DDL concurrency mode may be turned off with the set statement:

```
set ddl_concurrency off
```

In this case, the DDL statements will be part of the user's transaction. However, updates to the Ingres Star catalogs will cause exclusive locks to be held until the end of the user's transaction, thereby reducing multi-user concurrent access and increasing the chance of deadlocks.

Note: Turning off ddl_concurrency may force the transaction into a two-phase commit transaction if the user's transaction includes an update or a different node from the DDL statement. Such cases may be aborted by the server if any of the participating local databases do not support the slave two-phase commit protocol.

For a discussion of ddl_concurrency mode and rollback, see Rolling Back Transactions (see page 86).

Chapter 5: Using a Distributed Database

This chapter covers statements you use when you want to:

- Connect directly to a local database in "pass-through" mode or send a local DBMS-specific statement directly to the local DBMS
- Unload and reload a database
- Copy objects out of a database and restore it
- Modify catalogs
- Update catalog information
- Roll back transactions
- Request information from a database
- Get help on the registered names in your distributed database and their underlying local database objects (tables, views, and indexes)

This chapter tells you how to use Ingres Star with:

- direct connect
- direct disconnect
- direct execute immediate
- unloaddb
- copydb
- Sysmod
- verifydb
- rollback
- dbmsinfo( )
- help register

You use your distributed database the same way you use a single local database. There are, however, some statements you can use only with a distributed database and some statements you cannot use with a distributed database. For a summary of both types of statements, see the appendix "SQL Statement Summary."

Connecting Directly to a Local Database

While using Ingres Star, you can access an Ingres or Enterprise Access local DBMS directly. This is useful in some processing situations, for example, if you want to modify a table's storage structure, create secondary indexes, or grant other users access to a table.

Once you are in this directly connected or pass-through mode, Ingres Star merely passes through all statements and returns all responses. Statements are sent to the local database, whose server accepts or rejects the statement. When in this pass-through mode, Ingres Star does no syntax checking on the statement.

You can access a local DBMS directly with either of the following statements:

- direct connect
- direct execute immediate

Direct Connect Statement

The direct connect statement allows you to connect to a local database using Ingres Star in a pass-through mode. You remain in this pass-through mode until you issue a direct disconnect statement.

When you issue a direct connect statement, the Communications Server determines the login account on the remote node connection based on the netutil entries that have been set up. On a direct connect statement, your connection to the local DBMS will be made as the user authorized to access the remote node by netutil. For instance, if user Harry has a private authorization entry that defines Sally as the user for connections to node Italy, when Harry establishes a connection to node Italy from a distributed database, the local user will be Sally. This behavior also occurs when Ingres Star accesses a remote database's data or catalogs.

The direct connect does not always require that a new connection be opened between Ingres Star and the local DBMS. If the session has already caused Ingres Star to open a connection with the local DBMS, then Ingres Star uses that same connection for the direct connect.

You can connect to an Ingres or an Enterprise Access local DBMS with this statement, but not to another Ingres Star DBMS.

If you are within a transaction, you first must commit or rollback your transaction before you can issue a direct connect.

Direct Connect Syntax

The direct connect statement has the following format:

```
direct connect
       [with
       [node = node_name,
       database = database_name]
       [, dbms = server_class]]
```

node = *node_name*

Specifies the Net defined vnode name (the virtual node name) of the remote node that holds the database to which you want to connect. The default node is the current node. It may be delimited with double quotes.

If you specify the node = clause, you also must specify the database = clause.

database = *database_name*

Is the name of the local database to which you want to connect. The default database is the coordinator database. It may be delimited with double quotes.

If you specify the database = clause, you also must specify the node = clause.

The default for database and node is the coordinator database on the current node.

dbms = *server_class*

Is the type of local DBMS that contains the local database. It may be delimited with double quotes.

The server_class must be Ingres or one of the SQL Enterprise Access products. If you do not specify a server class, the default is the value in default_server_class on the remote installation (Ingres, unless defined otherwise.) To view or change this value, use the Configure Name Server screen of the Configuration-By-Forms utility.

Direct Disconnect Statement

The direct disconnect statement enables you to leave the pass-through mode enabled by your previous direct connect.

If you use direct disconnect before committing an active local DBMS SQL transaction, Ingres Star commits the transaction and sends the commit to the local DBMS.

Direct disconnect does not close Ingres Star's session with the local DBMS. Any state that you set up while directly connected remains in place and may cause side effects in Ingres Star's session with the local DBMS. Therefore, you must reset any session parameters (such as set statements) established during direct connect mode before issuing the direct disconnect. For example, if you issue a set autocommit on statement during a direct connect session, you must issue a set autocommit off before you issue the direct disconnect.

Direct Disconnect Syntax

The direct disconnect statement has the following format:

```
direct disconnect
```

Example: Direct Connect and Direct Disconnect

To place an integrity on an Ingres Star-level table, use the direct connect statement to define the integrity at the local level since an Ingres Star-level table actually resides in a local database.

```
create table employee (name char(100),
 dept integer, salary money)
   with node=remote1, database=mydb;
commit;
direct connect with node=remote1,
 database=mydb;
create integrity on employee is salary>0;
direct disconnect;
```

Direct Execute Immediate Statement

Use the direct execute immediate statement to send a local DBMS-specific statement to the local DBMS.

Ingres Star assumes that the statement being sent is an update operation to the local database. Ingres Star uses the two-phase commit protocol if the transaction involves an update to at least one other site.

The direct execute immediate statement has the following format:

```
direct execute immediate 'string_constant'
 [with
 [node = node_name,
 database = database_name]
 [, dbms =server_class]]
```

node = *node_name*

The Net defined vnode name (the virtual node name) of the remote node that holds the local database to which you want to connect. The default node is the current node. It may be delimited with double quotes.

If you specify the node = clause, you also must specify the database = clause.

database = *database_name*

The name of the local database to which you want to connect. The default database is the coordinator database. It may be delimited with double quotes.

If you specify the database = clause, you also must specify the node = clause.

The default for the database and node is the coordinator database on the current node.

dbms = *server_class*

The type of local DBMS that contains the local database. It may be delimited with double quotes.

The server_class must be Ingres or one of the SQL Enterprise Access products. If you do not specify a server class the default is the value in default_server_class on the remote installation (Ingres, unless defined otherwise.) Use the Configure Name Server screen of the CBF utility to view or change this value.

The with clause enables you to specify the node, database, and type of server to which you want to connect. No other with clauses are allowed when presented to Ingres Star.

Example: Direct Execute Immediate

The following example illustrates how you send a create integrity statement to a local DBMS to be executed using direct execute immediate:

```
create table employee (name char(100),
 dept integer, salary money)
   with node=remote1, database=mydb;
direct execute immediate 'create integrity
 on employee is salary>0'
   with node=remote1, database=mydb;
```

Direct Execute Immediate Statement Process

When the direct execute immediate statement is presented to Ingres Star, it strips off the direct keyword, the with clauses, and the quotes around the *string_constant* and sends the following statement to the local DBMS:

```
execute immediate string_constant
```

The local DBMS, possibly an Enterprise Access, strips off the execute immediate and then parses the query represented by *string_constant*.

If the query is a legal query for the execute immediate statement, it is executed. If the query is illegal, an error is returned.

Illegal Direct Execute Immediate Statements

The following statements are *not* allowed for direct execute immediate:

- Preprocessor directives:
  begin declare section
  declare
  end declare section
  include
  whenever
- Cursor statements:
  open
  close
  fetch
- Row returning statements:
  select
  endselect
- Non-database statements:
  call
  inquire_sql
  set_sql
- Transaction statements:
  commit
  rollback
  savepoint
  set autocommit on
- Other statements:
  connect
  disconnect
  describe
  direct connect
  direct disconnect
  execute
  execute immediate
  help
  prepare
  repeat queries
  register as link

Avoiding Execute Errors During Two-Phase Commit

It has already been pointed out that the transaction statements commit and rollback are illegal to use for direct execution in an Ingres Star session. These statements must not be passed through Ingres Star for execution on a remote non-distributed database or Enterprise Access because these statements can interfere with two-phase commits. This means that you must not attempt to implement remotely through Ingres Star:

- A direct execute immediate statement to execute commit or rollback
- A registered procedure, or a direct execute immediate statement to execute a procedure, that contains commit or rollback

Ingres Star, under the latest release of Ingres, detects such errors and returns an error message without performing the operation. However, earlier versions of Ingres serving the remote non-distributed database or Enterprise Access may not detect such errors.

Important! *If a user's commit or rollback statement is passed through Ingres Star and executed by an earlier release of Ingres during two-phase commit, the two-phase commit protocol may be disrupted and could corrupt the database.*

Direct Connect and Direct Execute Immediate Compared

Although the direct connect and direct execute immediate statements both access local DBMSs, they operate in different ways.

The direct connect statement:

- Is an Ingres Star statement only
- Can execute queries that return rows (for example, any select query)
- Cannot be issued inside an Ingres Star-level transaction. You must first commit the transaction
- Can execute transaction statements, commit, rollback, abort

The direct execute statement:

- Is an Ingres Star and Enterprise Access statement. It is used by Enterprise Access products to allow direct access to the underlying DBMS.
- Cannot run queries that return rows. It cannot run a select query.
- Can be run inside an Ingres Star-level transaction
- Cannot execute transaction statements, commit, rollback, abort.

Unloading and Reloading a Database

Unloaddb creates command files enabling a DBA to completely unload and reload a database. Unloaddb works in the same way as copydb except that unloaddb unloads all objects in the database of which you are the DBA, not just the ones you own.

With unloaddb, you can completely unload a coordinator database and move it to a different place. Ingres Star catalogs can be replicated so that the distributed database is always accessible despite node failure. For a full explanation of unloaddb and all its flags, see the *Command Reference Guide*. This section explains how unloaddb functionality changes when used with Ingres Star.

The unloaddb command does not do the actual unloading but creates SQL commands in two files in the current directory:

- unload.ing, which contains SQL instructions to read sequentially through the database copying every table into its own file in the named directory
- reload.ing, which contains SQL instructions to reload the database with information contained in the files created by the unload.ing file

When you use unloaddb with a distributed database, you must use the command twice:

1. You use unloaddb against the distributed database.

   This unloads the registrations of locally stored tables and views and distributed view definitions. Unloaddb does not unload the actual tables that may be stored anywhere in the distributed environment, only their registrations. In this respect, the unloaddb command works differently when used against a distributed database as compared to a local database. Because of this, you do not need to run unload.ing. You need run only reload.ing against the newly created distributed database.

2. You use unloaddb against the coordinator database.

   This unloads the user tables and data and the front-end object catalogs.

   The tables that are unloaded are the tables resulting from a create table statement at the Ingres Star level when a local database is not specified for the table's storage and the table is therefore stored in the coordinator database.

Accordingly, using unloaddb twice, you do not get all the tables that Ingres Star points to. You get the tables in the coordinator database, the distributed views in the distributed database, and the registrations in the distributed database that are linked to all the locally stored tables that make up the distributed database, and the front-end catalogs.

Important! *Because unloaddb must be run on the distributed database and the coordinator database separately, be careful not to unload the coordinator database in the same location as the distributed database in order to prevent files from being overwritten.*

Example: Unloaddb

The following example shows unloaddb used against a distributed database named distdbname and a coordinator database named iidistdbname:

UNIX:

```
cd /usr/mydir/iidistdbname
unloaddb iidistdbname
unload.ing

cd /usr/mydir/distdbname
unloaddb distdbname/star

createdb newdistdbname/star

cd /usr/mydir/iidistdbname
```

Edit reload.ing to replace 'iidistdbname' by 'iinewdistdbname':

```
reload.ing

cd /usr/mydir/distdbname

reload.ing
```

VMS:

```
set def usr_disk:[mydir.iidistdbname]
unloaddb iidistdbname
@unload.ing

set def usr_disk:[mydir.distdbname]
unloaddb distdbname/star

createdb newdistdbname/star

set def usr_disk:[mydir.iidistdbname]
```

Edit reload.ing to replace 'iidistdbname' by 'iinewdistdbname':

```
@reload.ing

set def usr_disk:[mydir.distdbname]
```

Edit reload.ing to replace 'distdbname' by 'newdistdbname':

```
@reload.ing
```

Copying Objects Using copydb

The command copydb creates command files enabling you to copy objects owned by you out of a database and restore it. For a full explanation of copydb and all its flags, see the *Command Reference Guide.*

The copydb command does not copy the database but creates two SQL command files in the current directory for doing the actual copying:

- *copy.out*, which contains SQL instructions to copy all the tables owned by the user into files owned by the user in the named directory
- *copy.in*, which contains SQL instructions to copy the files into tables, create indexes, and perform modifications

The copydb command works slightly differently when you use it against a distributed database. If you run copydb against a distributed database, it only copies out registration statements of local tables and views registered in the distributed database and distributed view definitions.

The SQL instructions in *copy.out* copy data out into files that can be accessed by the SQL instructions in *copy.in*. Because copying the distributed database involves executing registration statements and not copying data, you need run only *copy.in*.

Example: Copydb

The following example shows copydb used against a distributed database named distdbname and a coordinator database named iidistdbname:

UNIX:

```
cd /usr/mydir/iidistdbname
copydb iidistdbname
sql iidistdbname <copy.out


cd /usr/mydir/distdbname

copydb distdbname/star

createdb newdistdbname/star

cd /usr/mydir/iidistdbname
```

Edit copy.in to replace 'iidistdbname' by 'iinewdistdbname':

```
sql iinewdistdbname <copy.in

cd /usr/mydir/distdbname
```

Edit copy.in to replace 'distdbname' by 'newdistdbname':

```
sql newdistdbname <copy.in
```

VMS:

```
set def usr_disk:[mydir.iidistdbname]
copydb iidistdbname
sql iidistdbname <copy.out

set def usr_disk:[mydir.distdbname]
copydb distdbname/star

createdb newdistdbname/star

set def usr_disk:[mydir.iidistdbname]
```

Edit copy.in to replace 'iidistdbname' by 'iinewdistdbname':

```
sql iinewdistdbname <copy.in

set def usr_disk:[mydir.distdbname]
```

Edit copy.in to replace 'distdbname' by 'newdistdbname':

```
sql newdistdbname <copy.in
```

Modifying Catalogs Using sysmod

The sysmod command is an Ingres utility to modify system catalogs to predetermined storage structures. Running sysmod can improve performance by reducing system catalog overflow pages caused by a lot of DDL activity. Only the DBA or the system administrator can run sysmod.

You may use sysmod on a distributed database. You can specify particular Ingres Star catalogs as sysmod targets, or default to modifying all system catalogs.

For flags and full syntax details of the sysmod command, see the *Command Reference Guide.*

Example: sysmod

As the database administrator, you modify all system catalogs in distributed database mystardb. (This is the most common usage of sysmod.) Note that you do not use the /star suffix on the distributed database name:

```
sysmod mystardb
```

As the database administrator, you modify the Ingres Star catalogs iidbcapabilities, iiddb_tree, and iiddb_objects in distributed database herstardb:

```
sysmod herstardb iidbcapabilities iiddb_tree iiddb_objects
```

As the database administrator, you modify just the catalogs for the Vision product on database mystardb:

```
sysmod mystar -f vision
```

Updating Catalog Information Using verifydb

The verifydb command is an Ingres utility to provide database management, supportability, and disaster recovery services.

This section discusses the use of the refresh_ldbs operation of verifydb to notify Ingres Star of remote database upgrades to which a distributed database already has registered objects. For a full explanation of verifydb and all its flags, see the *Command Reference* Guide.

The refresh_ldbs operation of verifydb allows you to update distributed database Ingres Star catalog information dealing with the version level of remote non-distributed databases that are upgraded after the remote non-distributed databases have been registered with the distributed database. You want the remote non-distributed database to reflect its latest upgrade version so that Ingres Star will accept all upgraded features available to the remote non-distributed database.

The refresh operation is performed by specifying the verifydb command with the -orefresh_ldbs parameter. This operation reads the standard catalogs of each remote database processed and makes sure that the Ingres Star catalog correctly reflects the iidbcapabilities levels of the various remote non-distributed databases that contain objects registered to the distributed database. It does so without changing the contents or semantics of the standard catalog interface.

It is recommended that you run this operation on a distributed database after you run upgradedb on any of the remote databases accessed by the distributed database. For details on running upgradedb, see the appendix "Release Compatibility."

Examples: verifydb

Run verifydb with the refresh_ldbs option on the mystar database to check whether or not the iiddb_ldb_dbcaps catalog correctly reflects the level of its registered databases. Log output is to the default verify log file.

```
verifydb -mreport -sdbname mystar -orefresh_ldbs
```

Run verifydb with the refresh_ldbs option on the mystar database, correcting any iiddb_ldb_dbcaps catalog entries that incorrectly reflect the level of its registered databases. Log output is to the alternate refreshldb.log log file.

```
verifydb -mrun -sdbname mystar -orefresh_ldbs -lfrefreshdb.log
```

Run verifydb with the refresh_ldbs option on all databases that you own, correcting any iiddb_ldb_dbcaps catalog entries that incorrectly reflect the level of its registered databases. Log output is to the alternate refreshldb.log log file.

```
verifydb -mrun -sdba -orefresh_ldbs -lfrefreshdb.log
```

Run verifydb with the refresh_ldbs option on all databases that user sue owns, correcting any iiddb_ldb_dbcaps catalog entries that incorrectly reflect the level of its registered databases. Log output is to the default verify log file.

```
verifydb -mrun -sdba -orefresh_ldbs -usue
```

Rolling Back Transactions

The rollback statement rolls back part or all of the current Ingres Star transaction. Rollback to a savepoint applies only to those Ingres Star transactions involving Ingres local databases.

If the data definition language (DDL) concurrency mode parameter ddl_concurrency is set on, and rollback is executed after a:

- create table, the table in the local database is destroyed, but the registration of the table in the distributed database remains. You should remove the registration.
- drop table, the table in the local database remains but the registration of the table in the distributed database has been removed. You should re-register the table.

If ddl_concurrency is set on, and rollback is executed after a:

- create view is executed within a transaction, the view is not destroyed. You should drop the view.
- drop view, the view remains destroyed. You should re-create the view.

The ddl_concurrency is on by default. To turn off distributed database concurrency mode, see DDL Concurrency Mode (see page 70).

For a discussion on distributed transactions, see the chapter "Understanding Ingres Star Architecture." For more information on the rollback statement, see the *SQL* Reference *Guide*.

dbmsinfo( ) Function—Request Information from a Database

The dbmsinfo( ) function is an SQL function that is used to request information from a database.

This function has the following syntax:

```
dbmsinfo (request_name)
```

Request names for the dbmsinfo function that are supported by Ingres Star are shown here:

Request Name	Response Description
autocommit_state	Returns 1 if autocommit is on; 0 if autocommit is off.
_bintim	Returns the current time and date in an internal format, represented as the number of seconds since January 1, 1970, 00:00:00 GMT.
_bio_cnt	Returns the number of I/Os made by Ingres Star and all currently connected local DBMSs. Ingres Star returns "0" for this argument.
_cpu_ms	Cpu time for session in milliseconds. Ingres Star returns 0 for this argument.
database	Returns the database name.
dba	User name of the distributed database owner.
dbms_bio	Returns the number of buffered I/O requests for all connected sessions. Ingres Star returns 0 for this argument.
dbms_cpu	Returns the cumulative CPU time for the local DBMS, in milliseconds, for all connected sessions. Ingres Star returns 0 for this argument.
dbms_dio	Returns the number of direct I/O requests for all connected sessions. Ingres Star returns 0 for this argument.
_dio_cnt	Disk I/O requests made by Ingres Star and all currently connected local DBMSs. Ingres Star returns 0 for this argument.
_et_sec	Returns the elapsed time for session, in seconds.
language	Returns the language used in the current session to display messages and prompts.

Request Name	Response Description
_pfault_cnt	Page fault count for Ingres Star and all currently connected local DBMSs. Ingres Star returns 0 for this argument.
query_language	Returns 'sql' or 'quel'.
server_class	Returns the class of local DBMS, for example 'ingres'.
terminal	Returns the terminal address.
transaction_state	Returns 1 if presently in a transaction, 0 if not.
username	User name of the client connected to Ingres Star.
_version	Current version number of the Ingres Star process.

These request names are case insensitive. When you run a query against an Ingres local DBMS, the dbmsinfo() function always returns a varchar(32) as the result. When you run a query against a Star Server, the dbmsinfo() function returns a varchar of the length of the response.

The following query returns a variable length string containing the answer, for example, 1:

```
select i=dbmsinfo('transaction_state');
```

For more information on dbmsinfo( ), see the *SQL Reference Guide*.

help register Statement—Get Help with Objects

The help register statement gives you information about:

- Registrations in the distributed database catalogs
- Mapping between the registered distributed database object names and the names of the underlying linked objects
- Linked underlying objects in their respective local databases

This statement is an Interactive SQL Terminal Monitor statement only.

The help register statement returns the following information about the registered object and its underlying object:

- Registered name
- Owner of the registered name
- Type of object
- Node name
- Database name
- Local object name
- Owner of the local object
- Local DBMS class
- SQL text of the registration

The help register statement has the following format:

```
help register object_name
```

object_name

Is a table, view, or database procedure.

For more information on the help register statement, see the *SQL Reference Guide*.

Chapter 6: Managing a Distributed Database with StarView

This chapter discusses how to use StarView, a simple forms-based utility, to manage your distributed databases.

StarView Capablilties

Using StarView you can:

- Display all your distributed databases
- Display the nodes, databases, and tables registered in a distributed database
- Display the local database objects that make up a distributed database
- Test the network connections to each node in a distributed database
- Register local tables and views in a distributed database without using the SQL register as link statement
- Remove database objects that you had previously registered in your distributed database

In addition, you can query databases from within StarView with direct access to:

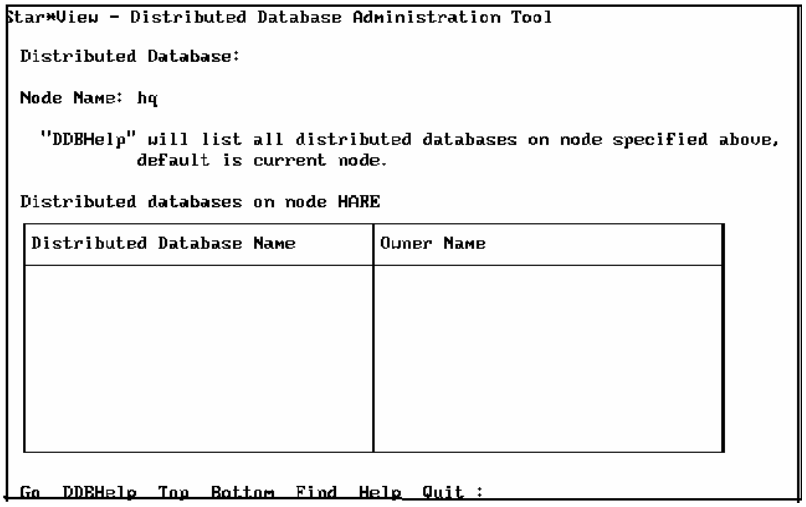
- Ingres Interactive SQL (ISQL)
- The Tables utility

Moving Around in StarView

The StarView utility operates in exactly the same way as Ingres forms-based tools. A brief summary of how to use forms follows. For a complete explanation of Ingres forms-based tools, see the *Forms-based Application Development Tools User Guide*.

Operations Menus

An operations menu is displayed at the bottom of each frame. The following figure shows a typical StarView frame:



You can cycle through all the menu selections by pressing the menu key repeatedly. (The key on your keyboard that acts as the menu key depends on your terminal and the individual key mappings you have chosen.)

To move the cursor from the window to the operations menu, press the menu key. To return to the window, press the Return key.

Long Operations Menus

Some operations menus are longer than the width of the frame. The presence of additional menu items is indicated by either a > at the right end of the menu, a < at the left end, or both.

You can cycle through the entire set of menu options by pressing the menu key repeatedly.

Moving Between Operations Menus

When you choose an operation from a menu, you often are presented with another frame containing a submenu of operations.

To return to the original menu, use the End operation. If you leave a submenu with the Quit operation, you quit StarView and return to the operating system prompt.

Options for Selecting an Operation

There are two ways to select an operation from the operations menu:

- Selection by function key
- Selection by name

Select an Operation by Function Key

To select an operation that has a function key (or key combination) mapped to it, simply press that key. This invokes the operation regardless of where the cursor is when you press the key. The function key that is mapped to an operation is shown in parentheses after the operation.

Select an Operation by Name

To select an operation from the menu by name, follow these steps:

1. Press the menu key.

   The cursor is moved to the menu.

2. Type either the full name or enough letters to uniquely identify the operation you want to select, and then press Return.

   For example, if the operations Find and Forget are displayed on the same menu line, you must type at least **fi** to identify Find and at least **fo** to identify Forget.

Context-sensitive Help

Context-sensitive help is available, based on the current task or field with which you are working.

You can obtain help by placing the cursor on the operations menu line and typing h for Help. You also can press the Help key to get help at any time. Sometimes, help is provided on several screens.

What to do

Describes the current screen and the operations menu

Keys

Describes the function and control keys and their current definition

Field

Displays a list of valid values for a field or the display format, data type, and validation check, if any, for a field

Help

Displays the type of Help available

End

Exits from any Help screen to the previous screen

To make a selection on a Help screen, type at least the first unique characters of the operation and press Return, or press the key listed in parentheses after the operation. To move through the Help screens, use the cursor movement keys specific to your terminal. You can see a list of the keys available by selecting the Keys operation from the Help menu.

Error Messages

You are provided with context-sensitive error messages that indicate both the error type and the error code.

Explanations for the errors are also provided. When you receive an error message, a single-line message is displayed with a prompt that tells you to press either the End key or the More key.

To see the explanation of the error, press the More key. After reading the explanation, press Return to return to your work in progress.

To exit the message without reading the explanation, press the End key at the prompt.

For additional information on error message formats, see the *System Administrator Guide*.

Start StarView

To call StarView type starview, or starview and a distributed database name, at the operating system prompt. You may also include a remote vnode name to run StarView against a distributed database on a remote node.

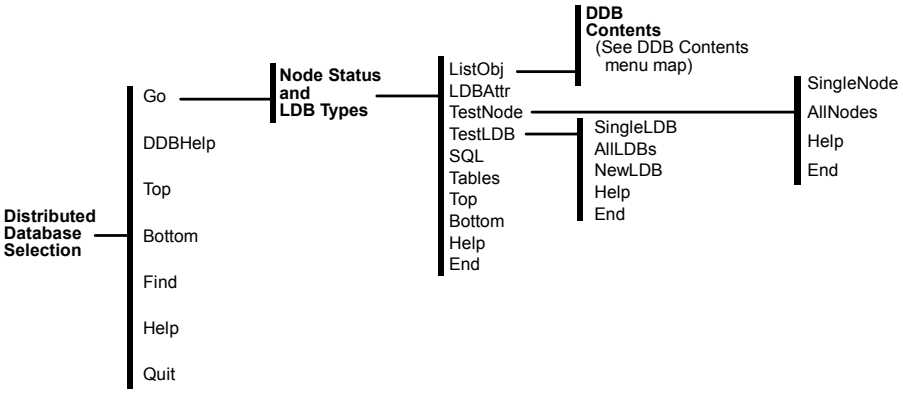
The complete syntax for StarView is:

```
starview [vnode::] [distdbname][/star]
```

- If you invoke StarView with a database name, the Node Status and LDB Types frame is displayed.
- If you invoke StarView without specifying a database name, the opening StarView main frame is displayed.

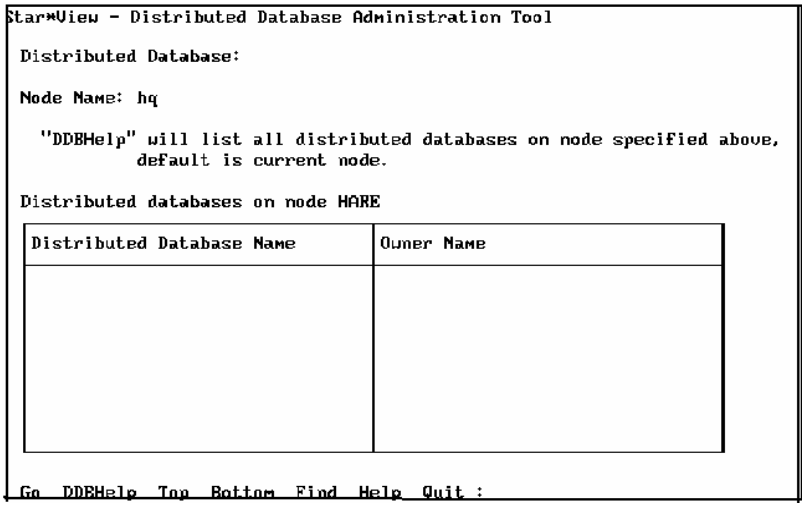
StarView Menu Map

The figures below show the menu selections available through the StarView utility:



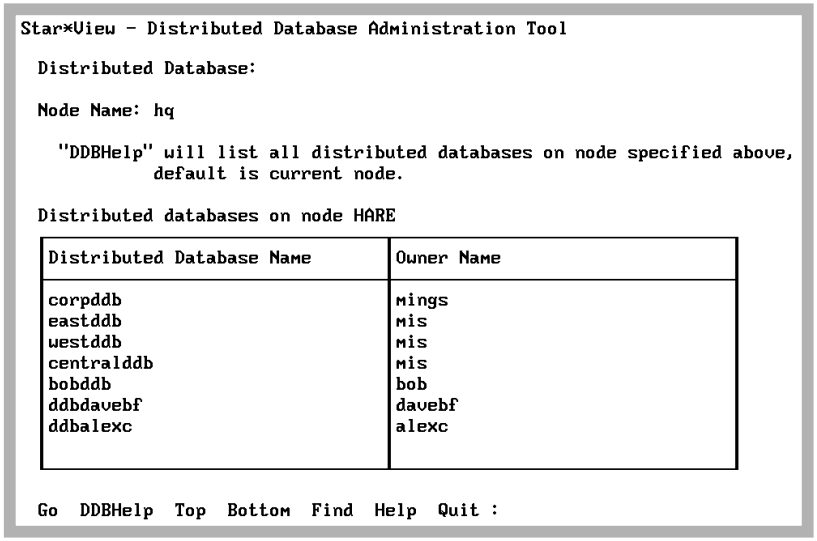
The DDB Contents Map

The following figure shows the menu selections available from the DDB Contents frame:



The StarView Main Frame

The StarView main frame is displayed in the following figure:



You can view and choose from a list of distributed databases or type the name of a distributed database. See Select a Distributed Database (see page 97).

The StarView main frame includes the following operations:

Go

Connects you with the displayed distributed database and displays its associated nodes and component local databases.

DDBHelp

Lists all distributed databases on the current node so you can select one. Your selection is automatically entered in the Distributed Database field.

Top, Bottom, Find, Help, Quit

These are the standard menu operations.

Select a Distributed Database

To select a distributed database, either select from a list of distributed databases or manually enter a distributed database name.

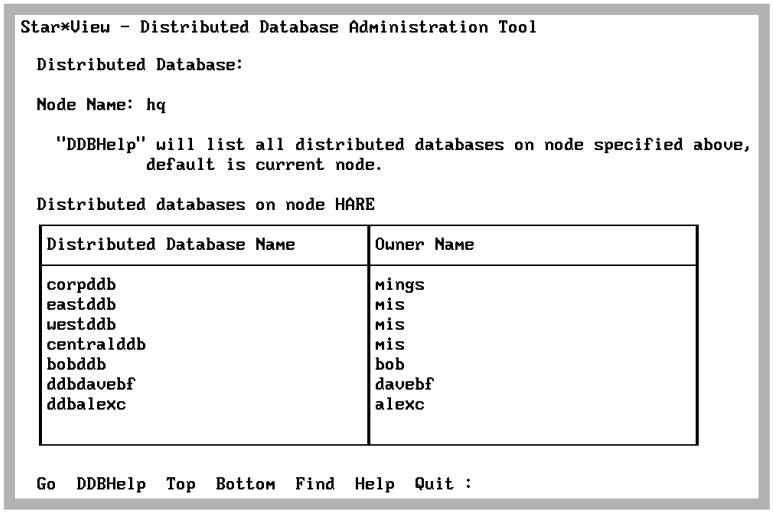
Note: If you specify a distributed database name on the StarView command line, this menu is skipped.

Select a Distributed Database from a List

To select from a list of distributed databases:

1. Choose DDBHelp.

   A list of distributed databases is displayed as shown in the following figure:



2. Move the cursor down.

   As the cursor moves, successive distributed databases are highlighted. Choose one.

3. Choose Go.

   You are connected to the selected database.

Select a Distributed Database by Entering a Name

To manually enter a distributed database name:

1. Move the cursor to the Distributed Database field and type in the desired distributed database name.
2. Choose Go.

   You are connected to the specified database.

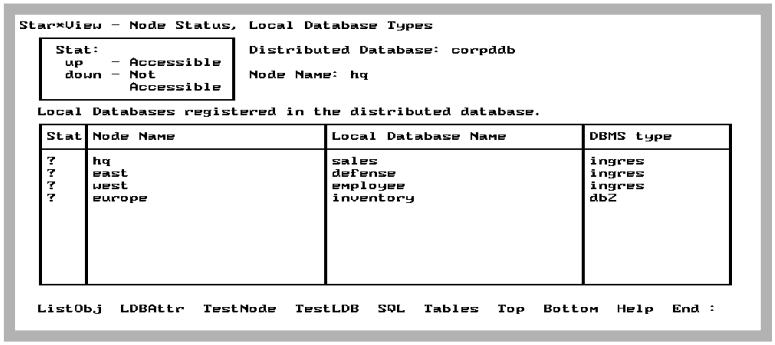
Node Status and Local Database Types Frame

Choosing the Go operation from the StarView main frame displays the Node Status and Local Database (LDB) Types frame.

The Node Status frame also displays first if you specify the name of a distributed database as a parameter when you type starview at the operating system command line.

To connect to another distributed database, you must choose End to call up the main StarView frame and Select another distributed database.

The following figure shows the Node Status and Local Database Types frame:



The Node Status and Local Database Types frame includes the following operations. Note that both SQL and Tables selections connect to the selected distributed database:

ListObj

Lists the objects in the selected distributed database.

LDBAttr

Lists the attributes of the selected local database.

TestNode

Tests whether you can connect to a node. You can test the accessibility status of a single node or all nodes.

TestLDB

Tests whether you can connect to a local database. You can test connections to a single database or all databases in the distributed database.

SQL

Accesses Interactive SQL (ISQL). You can enter and execute SQL statements at the interactive Terminal Monitor.

Tables

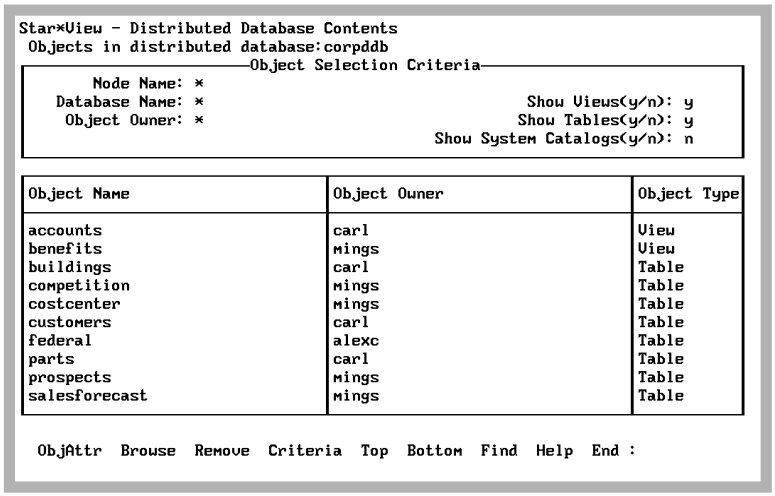
Accesses the Tables utility. You can access, create, destroy, and query all the tables in the distributed database using the forms-based Tables utility.

Top, Bottom, Find, Help, End

Standard menu operations.

ListObj Operation

The ListObj operation displays the Distributed Database Contents frame, which lists the component objects of the distributed database and the system catalogs:

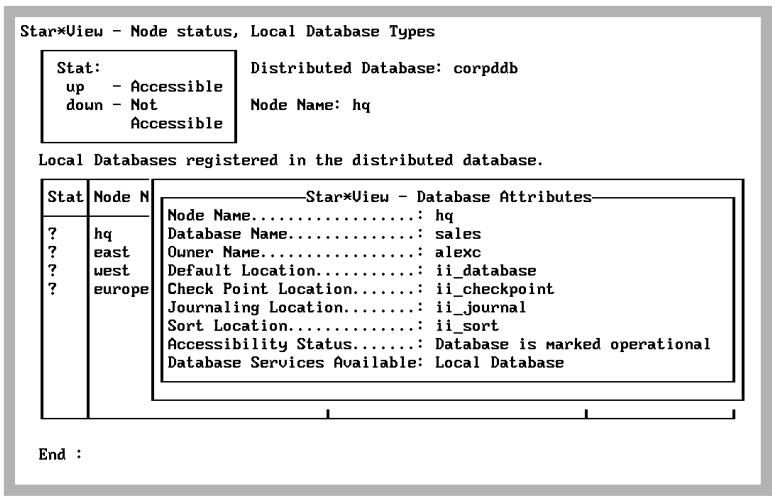


You can display all or any combination of these objects by altering the default settings.

To learn how to restrict the display of this frame to, for example, tables only, see Distributed Database Contents Frame (see page 104).

LDBAttr Operation

The LDBAttr (local database attributes) operation displays the following frame showing the attributes of your selected local database:



The attributes shown include name, date, checkpoints and journaling.

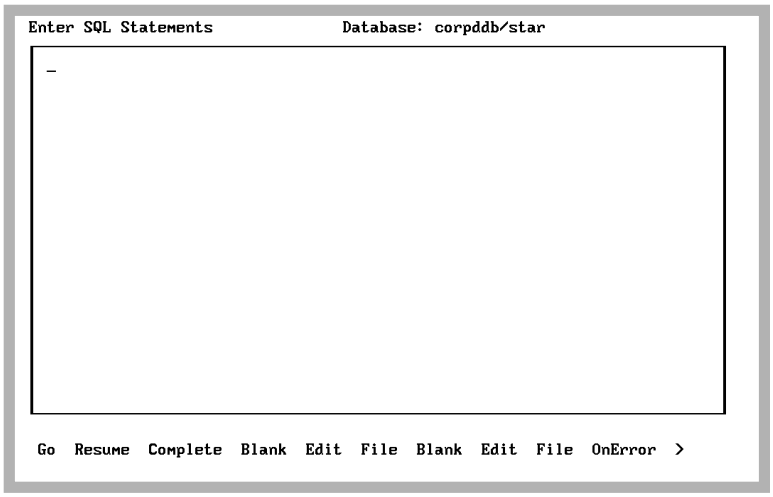
The Accessibility Status field displays one of the following messages:

- Database is marked operational
- Database is being destroyed
- Database is being created

The Database Service Available field displays either Local Database or Coordinator Database.

SQL Operation

By choosing the SQL operation, you can use Interactive SQL (ISQL) to query your distributed databases. The following screen is displayed:

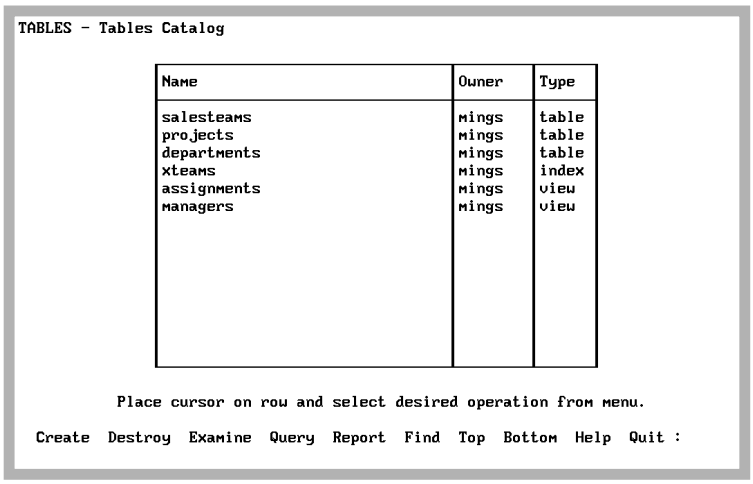


For a full explanation of ISQL, see *Forms-based Application Development Tools User Guide.*

Tables Operation

By choosing the Tables operation, you can use the forms-based Tables utility to query tables in your distributed databases.

The following screen is displayed:



If you create any tables using the Tables utility, the underlying local tables are created in the Coordinator database.

For a full explanation of the Tables utility, see *Forms-based Application Development Tools User Guide.*

TestNode Operation

By choosing the TestNode operation, you can test the node connections within your distributed database. When you test a node, the Status field on the frame changes from a question mark (?) to either up (accessible) or down (not accessible).

You can test the connections to a single node or to all nodes. The following list of options is displayed:

SingleNode

Test a single node

AllNodes

Test all the nodes in the distributed database

Help and End

Standard menu operations

TestLDB Operation

By choosing the TestLDB operation, you can test the availability of the local databases in your distributed database. When you test a database, the Status field changes from a question mark (?) to either up (accessible) or down (not accessible).

You can test the connections to a single database or to all databases. After testing one database, you can then test another database by choosing NewLDB.

When you choose AllLDBs, StarView goes through each local database and changes the display of the status to either up or down.

The following list of options is displayed:

SingleLDB

Test a single local database

AllLDBs

Test all the local databases in the distributed database

NewLDB

Test a new local database

Help and End

Standard menu operations

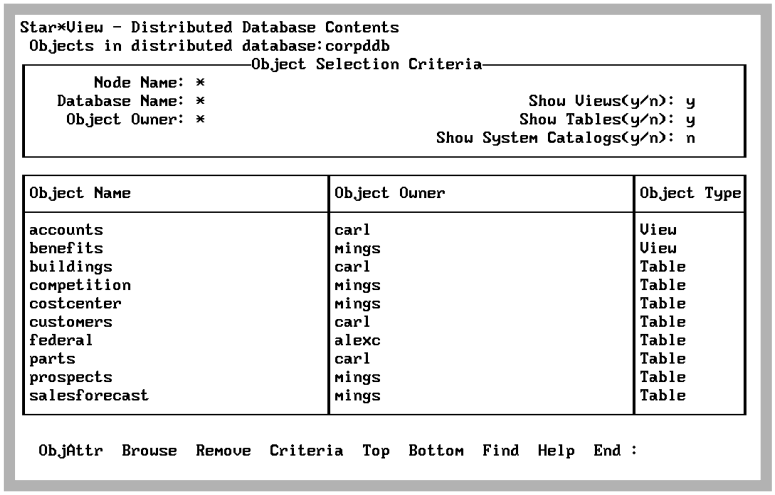
TestNode versus TestLDB

TestNode allows you to test whether you can connect to a node. TestLDB allows you to test whether you can connect to a local database.

If you have many local databases and only a few nodes, it is faster to test connections against the nodes rather than test every local database on the node.

Distributed Database Contents Frame

To see all the objects in your distributed database, choose the ListObj operation from the Node Status and Local Database Types frame. The Distributed Database Contents frame is displayed:



All the tables, views and indexes in your distributed database are listed by name, owner and type of object.

Registered indexes will have a name determined by Ingres Star at registration time of the form:

ddx_nnnn_nnnn

For example:

Object Name	Object Owner	Object Type
ddx_1323_1324	$ingres	Index

Note: Ingres Star does not retain any index names created when the index was created in the local database.

You can restrict the objects displayed in this frame by choosing the Criteria menu item. The criteria that control the Distributed Database Contents frame are described below:

Node Name

The node containing the local database. This can be modified by the NodeHelp menu item of the Criteria menu.

Database Name

The local database name. This can be modified by the LDBHelp menu item of the Criteria menu.

Object Owner

The owner of the object in the database. This can be modified by the OwnerHelp menu item of the Criteria menu.

Show Views

When y is selected all views in the distributed database are displayed.

Show Tables

When y is selected all tables in the distributed database are displayed.

Show System Catalogs

When y is selected all system catalogs are displayed.

When you choose the ListObj operation the first time, the following default criteria for the display of the objects in the distributed database are set:

Field	Default
Show Views	y
Show Tables	y
Show System Catalogs	n

By changing these values, you can restrict or extend the display of the objects in your distributed database. For example, by setting Show Tables to n, no tables are displayed.

To change the defaults, you must select the Criteria operation. A pop-up window is displayed on the DDB Contents frame within which you can amend the criteria fields. For details, see The Criteria Operation (see page 112).

DDB Contents Frame Operations

The Distributed Database Contents frame includes the following operations:

ObjAttr

Provides attribute information about the objects in the selected distributed database.

Browse

Allows you to look at a list of nodes and select a node. This list of nodes includes only those already used in the distributed database selected.

From the selected node, you may look at and select a local database. From the selected database, you may look at the tables and views in that local database. You may then select and register a table or view in your distributed database. (Registering Tables with StarView.)

You cannot use StarView to add a table from a node not yet in that distributed database (since StarView obtains the information from the iiregistered_objects catalog).

Remove

Removes the registration of tables or views from the selected distributed database (it does not drop the underlying table or view from the local database).

You cannot remove a table or view only a registration. To remove a table or view, you must exit StarView and use the drop statement.

Criteria

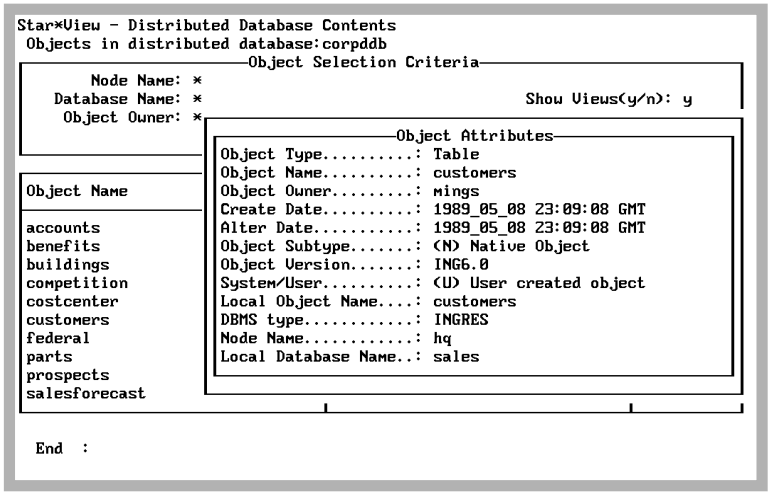
Restricts or extends the display of distributed database objects.

Top, Bottom, Find, Help, End

Standard menu operations.

The ObjAttr Operation

To show information about any object displayed in the Distributed Database Contents frame, select an object and choose the ObjAttr operation. The Object Atttributes pop-up window is displayed:



The following attributes of the chosen table or view are displayed in the pop-up window:

Object Type

The type of object: table, a view, or an index

Object Name

Name of the object

Owner

Owner of the object

Creation Date

Date the object was created

Alter Date

Date the object was last altered

Object Subtype

The subtype of the object, either Native (created at the Ingres Star level) or Link (created locally and registered as link)

System/User

Indicates whether the object is user-created or system-created

Local Object Name

Local name of the object

DBMS type

Type of local DBMS

Node Name

Name of the node

Local Database Name

Name of the local database

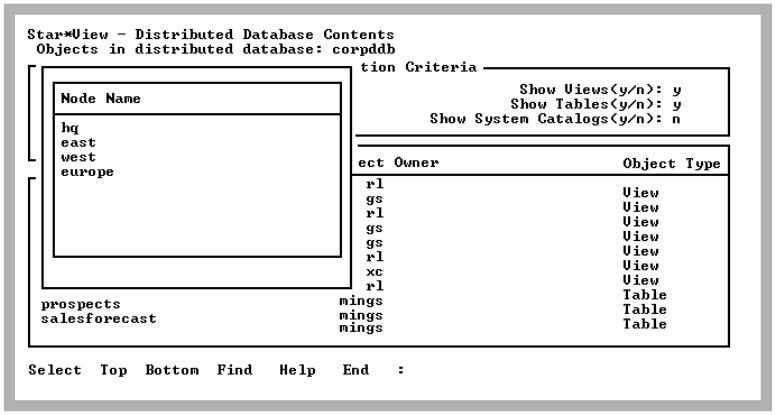
Depending on the type of object selected, the pop-up window displays information about a local table, view, or index, or a distributed view.

The Browse Operation

To see all the components of a distributed database, use the Browse operation. You may view the component nodes, the databases on that node, and finally the objects such as tables and views within any of those databases. When the objects within a local database are displayed, you may register them in your distributed database.

Choose the Browse operation from the Distributed Database Contents frame.

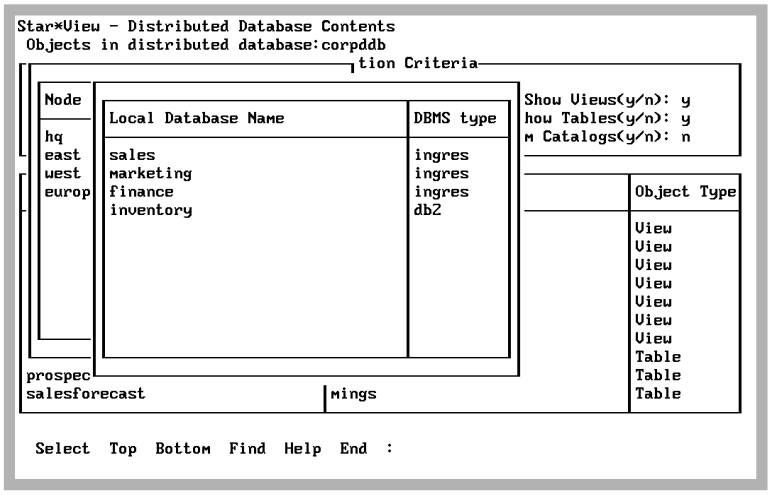
The following pop-up window is displayed:



All nodes are listed in the pop-up window.

To see all the databases on a particular node, select the desired node.

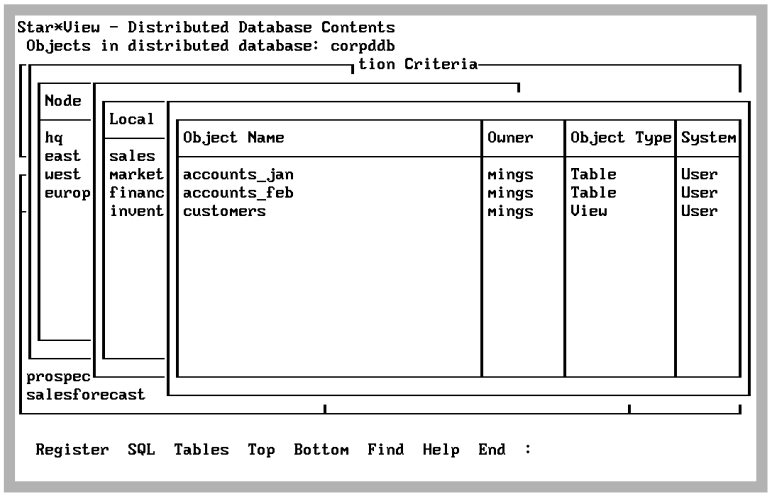
Move the cursor to the desired node and choose Select. The Local Database Name pop-up window is displayed:



All the databases on your selected node are displayed.

To see all the tables and views in a particular database, select the desired database.

Move the cursor to the desired database and choose Select. The Object Name pop-up window is displayed:



All the objects in your selected database are displayed. From this window, you can register tables and views in your distributed database. For details on registering an object in a distributed database, see Register Tables with StarView (see page 115).

Using the SQL operation, you can access Interactive SQL. Using the Tables operation, you can access the Tables utility. Both SQL and Tables connect to the selected local database. For a full explanation of ISQL and the Tables utility, see *Forms-based Application Development Tools User Guide.*

The Remove Operation

You can delete registrations from your distributed database using the Remove operation.

Removing the registration of a table removes the registered table name from the distributed database. It does not remove the underlying table or its data from the local database.

Removing the registration of a local view removes the registered view name from the distributed database. The underlying view, tables and data are not affected.

You may *not* delete a local table or a local or distributed view displayed in the list of objects in the distributed database. To delete the actual table or view, you must exit StarView and use the drop statement.

You may *not* use the Remove operation against an index or a distributed view.

Remove a Registration

To remove a registration from your distributed database:

1. From the Node Status and Local Database Types frame, choose ListObj from the menu.

   The Distributed Database Contents frame is displayed.

2. Move the cursor to the registered table or view you wish to remove.

   The object is highlighted.

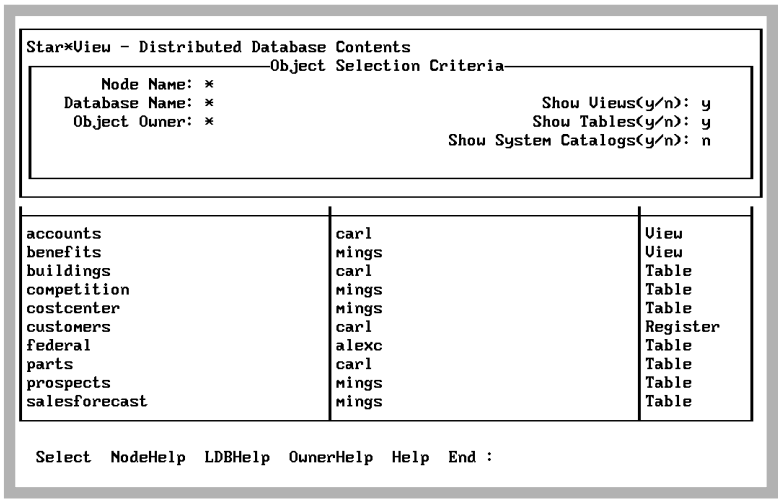
3. Choose Remove from the menu.

   The table or view is removed from the distributed database.

The Criteria Operation

You can restrict or extend the display of the objects in your distributed database with the Criteria operation. For example, if you change the Show System Catalogs to y, system catalogs are also displayed.

To set criteria, choose the Criteria operation from the Distributed Database Contents frame. The following pop-up window is displayed within which you can amend the criteria fields:



The cursor appears on the first field (Show Views). Move to the field you wish and change the criteria to y or n. After making your selection, choose Select.

The Node Name, Database Name, and Object Owner fields are modified, respectively, by the Nodehelp, LDBHelp, and OwnerHelp menu items.

The following operations are available from the Criteria pop-up window:

Select

Changes the object selection criteria to current values and returns to the Distributed Database Contents frame. That frame now displays only the objects adhering to the newly selected criteria.

NodeHelp

Lists the nodes in the distributed database for selection into the Node Name field.

LDBHelp

Lists the local databases for selection into the Database Name field.

OwnerHelp

Lists the owners for selection into the Object Owner field.

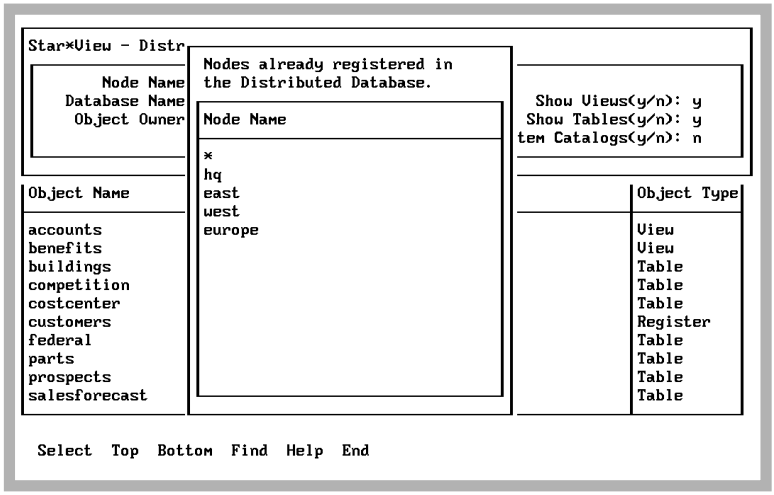
Help and End

Standard menu operations.

The NodeHelp Operation

To place a node name in the Node Name field, choose the NodeHelp Operation from the Object Selection Criteria frame.

The following pop-up window displays the nodes registered in your distributed database:

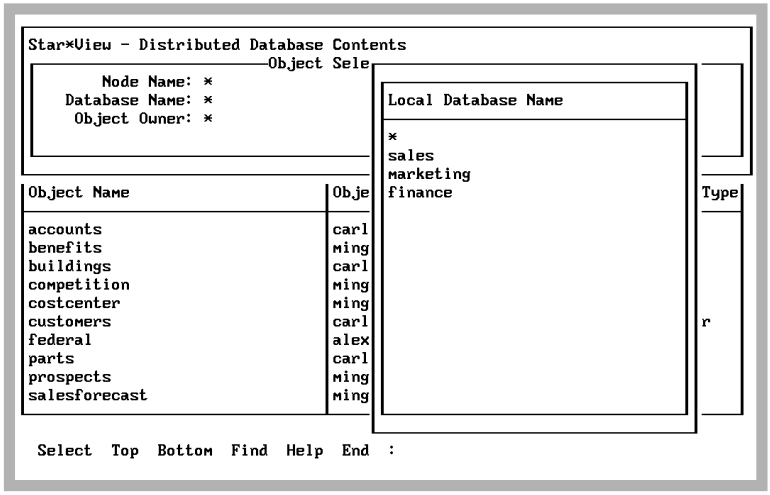


To choose a node name, place the cursor on a node name and choose Select. The node name is selected and placed in the Node Name field in the Object Selection Criteria Window.

The LDBHelp Operation

To place a database name in the Database Name field, choose the LDBHelp operation from the Object Selection Criteria frame.

The following pop-up window displays a list of local database names registered in the distributed database:

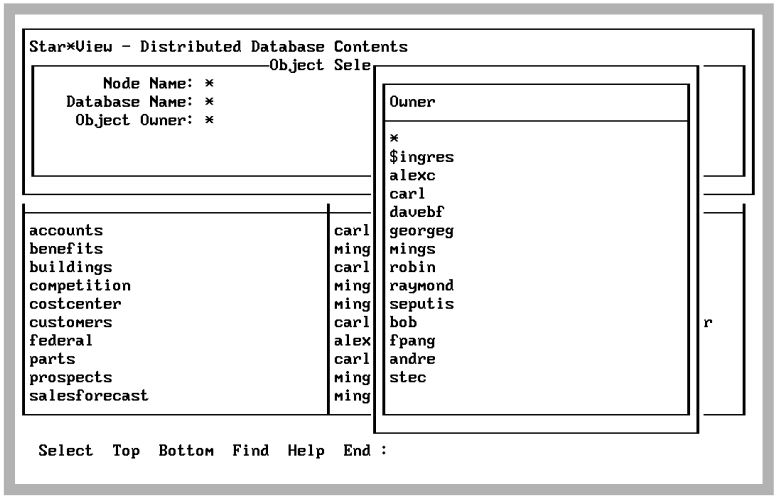


To select a database name, place the cursor on a database name and choose Select. The database name is placed in the Database Name field.

The OwnerHelp Operation

To place an owner's name in the Object Owner field, choose the OwnerHelp operation from the Object Selection Criteria frame.

The following pop-up window displays a list of owners of objects in local databases in the distributed database:



To select an owner name, place the cursor on an owner name and choose Select. The owner name is placed in the Object Owner field.

Register Tables with StarView

Populate a distributed database with tables that already exist in local databases by registering those tables in your distributed database. To do this, use the Register as Link Statement (see page 46). However, StarView provides an easy way to search for, select, and register local tables in your distributed database.

StarView also allows you to delete the registration of these tables using the Remove operation. Removing the registration of a table from your distributed database does not affect the underlying tables in the local database. To delete the table from a local database, you must quit StarView and use the drop statement.

Register Tables in a Distributed Database

To register tables in your distributed database, follow these steps:

1. At the operating system command line, type **starview**.

   The StarView main frame is displayed.

2. Select a distributed database, and then choose the Go operation.

   The Node Status and Local Database Types frame is displayed.

   As an alternative to Steps 1, 2, and 3, at the operating system command line type **starview** *distdbname*. The Node Status and Local Database Types frame is displayed.

3. From the Node Status and Local Database Types frame menu, choose the ListObj operation.

   The Distributed Database Contents frame is displayed.

4. From the Distributed Database Contents frame menu, choose the Browse operation.

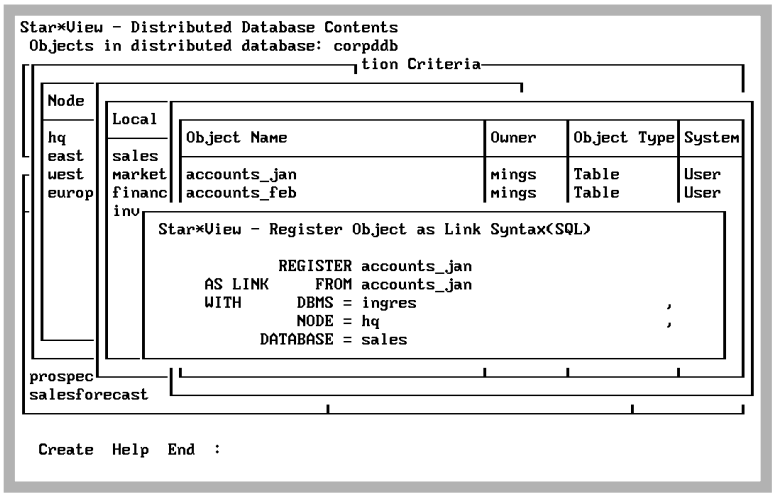
   A pop-up window listing the nodes is displayed.

5. Highlight your desired node and choose the Select operation.

   A pop-up window listing the databases on your selected node is displayed.

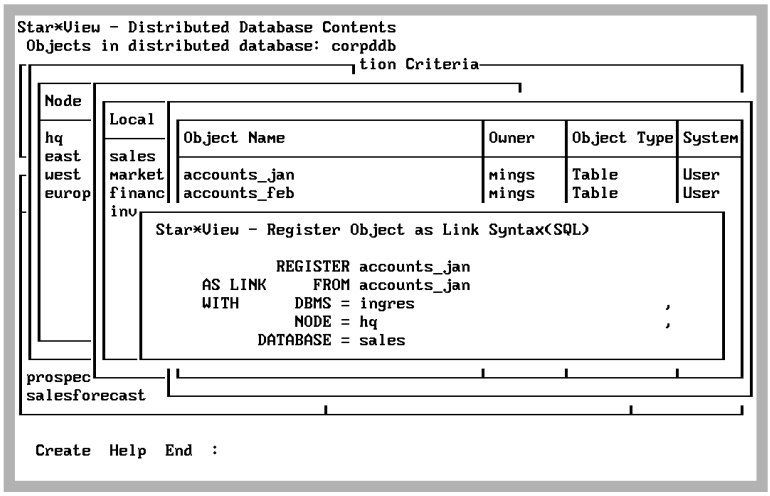
6. Highlight your desired database and choose the Select operation.

   The pop-up window listing all the objects in your selected database is displayed:



7. Highlight your desired table and choose the Register operation.

   The following Register Object as Link Syntax pop-up window is displayed:



If an object is highlighted when you choose the Register operation, the REGISTER, FROM, NODE, and DATABASE fields are completed for you automatically by StarView.

To register a table with a different registered name to its local name, enter the name in the REGISTER field of the Register Object as Link Syntax window and choose Create from the menu.

To register a table with a registered name that is the same as its local name, simply choose Create from the menu.

Register Other Database Objects

To register other types of local database objects such as views in your distributed database, you follow exactly the same procedure as for registering tables.

Chapter 7: Understanding Ingres Star Catalogs

This chapter describes the Ingres Star system catalogs.

Ingres Star Catalogs

Ingres Star catalogs consist of database tables that describe the objects in the distributed database. They are maintained to keep track of these objects. They are primarily for Ingres Star's own use, but you can use them in programs and applications to access (but not update) information about the distributed database.

Each catalog has columns—or attributes—with specific database management functions and rows that reflect different aspects of the database.

There are three types of Ingres Star catalogs:

- iidbdb catalogs
- Standard catalogs
- System catalogs

iidbdb Catalogs

Ingres Star uses four catalogs in the iidbdb:

- iidatabase
- iistar_cdbs
- iistar_cdbinfo
- iiddb_netcost

The iidatabase Catalog

The iidatabase catalog in the iidbdb is used to determine if a given database exists in the installation. This catalog has a column *dbservice* that is used to determine whether or not the given database is distributed, and in the case of a local database, whether it is a coordinator database.

The iistar_cdbs Catalog

The iistar_cdbs catalog in the iidbdb is used to store the identities and locations of coordinator databases associated with each distributed database.

This catalog contains an entry for the coordinator database associated with each distributed database. It is used by Ingres Star to determine the identity and residence of the associated coordinator database when a distributed database is invoked.

Column Name	Data Type	Description
ddb_name	char(32)	Name of distributed database
ddb_owner	char(32)	Owner of distributed database
cdb_name	char(32)	Name of coordinator database
cdb_node	char(32)	Name of the coordinator database's node
cdb_owner	char(32)	Owner of the coordinator database
cdb_dbms	char(32)	Server of coordinator database, for example, INGRES, DB2
schema_desc	char(32)	Reserved for future use
create_date	char(25)	Date when coordinator database was added
original	char(8)	Reserved for future use
cdb_id	integer4	Contains a unique database identifier corresponding to iidatabase.db_id for the coordinator database entry
cdb_capability	integer 4	Reserved for future use

The iistar_cdbinfo Catalog

The iistar_cdbinfo catalog provides maps between the distributed database and its underlying coordinator database. It indicates on which node the database was created, who owns it, and when it was created.

This catalog exists only in the iidbdb, not in all distributed databases. This catalog is read-only; you cannot update it.

Column Name	Data Type	Description
ddb_name	char(32)	Name of distributed database
ddb_owner	char(32)	Owner of distributed database
cdb_name	char(32)	Name of coordinator database
cdb_node	char(32)	Name of the coordinator database's node
cdb_owner	char(32)	Owner of the coordinator database
cdb_dbms	char(32)	Server of coordinator database, for example, INGRES, DB2
cdb_create_date	char(25)	Date when coordinator database was added

The iiddb_netcost Catalog

The iiddb_netcost catalog in the iidbdb is used to weigh the relative network costs of a transaction in order to compute the best query execution plan (QEP). Data in this catalog is used by Ingres Star's distributed optimizer.

Column Name	Data Type	Description
net_src	char(32)	Name of the source node
net_dest	char(32)	Name of the destination node
net_cost	float8	Cost of moving one byte from the source node to the destination node as a multiple of 1 DIO (Disk I/O). This field contains a float that is the cost of transferring one byte from the source to the destination site. This cost should be made in terms of DIO units. Network costs are added to DIO costs in order to determine which plan is cheapest. See the example below.
net_exp1	float8	Expansion field (should be set to zero)
net_exp2	float8	Expansion field (should be set to zero)

All data transfers are made by first transferring the data to the Star Server from the source site, then transferring data from the Star Server to the destination site. As a result, the only entries in the iiddb_netcost table that will be useful are those that include the Star Server node name as one of the sites.

Note: The StarView utility does not allow users to populate the iiddb_netcost catalog. If your configuration contains greatly differing network costs and you wish to provide network cost information to Ingres Star, you must do so manually.

To make inserts and updates into the iiddb_netcosts catalog, you must be a privileged user and log in as the installation owner. At the operating system prompt, enter:

UNIX:

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

VMS:

```
sql iidbdb -u$ingres "+U"
```

The Star Server must be restarted for the new iiddb_netcosts values to take effect.

When you make changes to the iiddb_netcost catalog, you can analyze differences in query plan strategies by using the set qep statement. The network or N costs are printed in the last line of each node in the query plan.

For further details on query execution plans, see the *Database Administrator Guide.*

Example: Net_cost

Assume that the Star Server is located on a node named *sanfrancisco* (using lower case is the default), and that the remote sites are named *newyork* and *washington*.

Some of the entries in iiddb_netcost could be:

net_src char(32)	net_dst char(32)	net_cost (f8)
sanfrancisco	newyork	0.001
sanfrancisco	washington	0.002
sanfrancisco	sanfrancisco	0.0002

Note that there is an entry in which *sanfrancisco* is both the source and destination sites. This represents transfers of data from the Star Server to and from local databases on the Star Server site (including the coordinator database), but these costs are relatively low.

Also, typically, the same cost applies for either direction so that if only one row exists between two nodes, the other direction is assumed to be the same cost.

For example, suppose 10000 bytes is to be transferred from *newyork* to *washington*. The data is routed through *sanfrancisco* since the Star Server exists on that node. The cost of transferring from *newyork* to *sanfrancisco* is 0.001*10000=10 units, and the cost from *sanfrancisco* to *washington* is 0.002*10000=20 units, so the total cost is 30 units. If there were 20 disk I/Os and 2 CPU units involved, the total cost would be 52 units.

Standard Catalogs

The standard catalogs let you get information from the system catalogs, which may not be queried directly. For details on the standard catalogs, see the appendix "Standard Catalog Interface."

Important! *Users and user applications may query the standard catalogs but may not update them*.

The standard catalogs provided with the current release are identical in specification and function to the Ingres 2.6 standard catalogs, except as noted in the following table:

Catalog Name	Exceptions to Release 2.6 Catalog
iialt_columns	None
iicolumns	None
iidbcapabilities	Has the additional entries: OWNER_NAME, STANDARD_CATALOG_LEVEL, OPEN/SQL_LEVEL, DB_DELIMITED_CASE, and DB_REAL_USER_CASE.
iidbconstants	Has the added column system_owner
iihistograms	None
iiindexes	None
iiindex_columns	None

Catalog Name	Exceptions to Release 2.6 Catalog
iiintegrities iimulti_locations iipermits	Not used. Ingres Star does not support permits or integrities. For multiple locations, partitioning across several locations is managed by the autonomous local database server. You may query these catalogs, but Ingres Star never populates them with data, so your answer will always be a null set of rows. If you want permit, integrity or location information about a registered object, do a direct connect to the appropriate local database and query the local database's standard catalogs.
iiphysical_tables	None
iiprocedures	None
iiregistrations	None
iiregistered_objects	An Ingres Star-only standard catalog
iistats	Has the added columns: column_domain, is_complete, stat_version, hist_data_length.
iitables	Column location name is 32 characters long, not 24.
iiviews	None

System Catalogs

System catalogs store specific information for Ingres Star. These catalogs are used to get the information needed to operate on distributed objects in the database. Users may not query these catalogs directly. The standard catalog interface described in the appendix, "Standard Catalog Interface" lets you access data from the system catalogs.

The table names of some Ingres Star system catalogs can be used as arguments to the sysmod command, but these tables are not supported for any other use. The following Ingres Star system catalogs are legal targets for sysmod:

Catalog	Description
iiddb_dbdepends	Dependency tree for a distributed view
iiddb_ldbids	Contains information about each local database known to the distributed database

Catalog	Description
iiddb_ldb_columns	Map of local table's column names if the table is registered with user-supplied aliases to the column names
iiddb_ldb_dbcaps	Contains capability data on each local database known to the distributed database
iiddb_long_ldbnames	Contains the full local database name (if it exceeds 32 characters) and the alias to Ingres Star's 32-character name
iiddb_object_base	Used to generate a unique identifier
iiddb_objects	Describes distributed objects known to the distributed database
iiddb_tableinfo	Data on the underlying (local database) objects for distributed objects
iiddb_tree	Used to store Ingres Star-generated trees, such as view definitions

The following Ingres Star system catalogs exist in a distributed database, but these catalogs are not legal targets for sysmod:

Catalog	Description
iiddb_dxldbs	List of local databases involved in a Ingres Star two-PC transaction
iiddb_dxlog	Log of an Ingres Star two-PC transaction
iiddb_xdxlog	Secondary index on iiddb_dxlog

Mapping Ingres Star Objects

Ingres Star recognizes and manages three types of queryable objects: tables, views, and indexes. It also maps procedures, which are not queryable but may be executed.

Registered Names and Related Catalogs

The basic function of a registered name is to allow a local queryable object to be given the status of a queryable object in the distributed database. The register as link statement does this by registering a name in the distributed database to denote a queryable object in a local database. This allows you to use a registered name to denote its underlying object.

To support registered name transparency, it is necessary for Ingres Star to promote the complete information about the local object from the local database catalogs into Ingres Star's equivalent catalogs and classify the information under the registered name and the owner of the registered name.

Tables and Related Catalogs

A table's local catalog information is promoted into the Ingres Star catalogs as follows:

- From the local database's iitables to Ingres Star's iitables
- From the local database's iicolumns to Ingres Star's iicolumns
- From the local database's iiphysical_tables to Ingres Star's iiphysical_tables
- From the local database's iiindex_columns to Ingres Star's iiindex_columns
- From the local database's iialt_columns to Ingres Star's iiialt_columns

Index Mapping

Each index of a local table introduced by a register is promoted as an index and given a registered name composed of the prefix ddx, the base, and the index of the object id, separated by the special character, _, for example, ddx_2002_2003.

Indexes are automatically mapped during the register statement to iiindexes and iiindex_columns. You cannot create secondary indexes with Ingres Star.

Table and Column Mapping

If no column mapping is specified during the register as link statement, data is promoted from the local database's iicolumns to Ingres Star's iicolumns.

- The local information is stored in the Ingres Star-specific catalog iiddb_ldb_columns. The representation in iiddb_ldb_columns is used exclusively by Ingres Star.
- The column information of the registered name is stored in Ingres Star's iicolumns standard catalog.

Ingres Star provides column mapping if either the distributed database or the local database (but not both) support mixed-case delimited identifiers.

Physical Information and Statistics Mapping

If the local object is an index or table (not a view), more relevant information is transferred:

- from the local database's iihistograms to Ingres Star's iihistograms
- from the local database's iistats to Ingres Star's iistats

This is promoted only if the environments are compatible, that is, the hardware and operating system are the same on the local database's node as on the distributed database's node.

Statistics information is not promoted if the local database and distributed database are in heterogeneous environments. To obtain statistics in such an environment, you must run optimizedb.

Local Tables Index Information

If the local table is in an Ingres database, each index is also registered. Note that Enterprise Access indexes might not be promoted to Ingres Star.

Views and Related Catalogs

When a view is created, its information is stored in several Ingres Star catalogs that include the following standard catalogs:

- iicolumns
- iitables
- iiviews
- iiphysical_tables

Registered Procedures and Related Catalogs

When Ingres Star registers procedures, it maps a local procedure and local owner name to a distributed procedure and distributed owner name. It does not map procedure parameters; it merely passes those through to the local DBMS when the procedure is executed.

Registered procedures are stored in the following standard catalogs:

- iiregistrations
- iiprocedures

Note: Registered procedures are not stored in the iitables standard catalog.

Appendix A: Release Compatibility

This appendix discusses the compatibilities of various releases of Ingres with the current release.

Note: All nodes accessed by Ingres Star must be running Release 6.4 or higher.

It also discusses using the upgradedb and upgradefe utilities with Ingres Star.

Utilities for Updating a Release 6.4 Star Database

Use the upgradedb and upgradefe utilities to update a Release 6.4 Star database to the current release.

- Upgradedb is a utility to reformat catalogs from Release 6.4 versions to the correct format for the current release. It upgrades DBMS, Standard, Ingres tools and Star catalogs to the current release. User tables are not affected by upgradedb.
- Upgradefe is a utility that will upgrade Ingres tools catalogs to the format required by the current release. It may also be used to install new tools clients. (Upgradedb uses upgradefe to update tools catalogs.)

The following sections discuss Ingres Star-specific usage of these conversion utilities. For complete details on using upgradedb and upgradefe, see the *Migration Guide.*

Ingres Star and Upgradedb

Upgradedb converts distributed databases. You can specify the distributed database name with or without the /star flag.

You can run upgradedb regardless of whether the Star Server is running. However, if you use upgradedb without a Star Server, all tools catalog upgrades are skipped, and you must later run upgradefe through a Star Server to upgrade the tools catalogs.

Upgradedb cannot be used on coordinator databases.

Upgradedb Command—Update a Distributed Database

The upgradedb command for distributed databases has the following format:

```
upgradedb distdbname [/star] | -all [-f product {product}]
```

distdbname

Specifies the name of the distributed database to be converted.

/star

(Optional) Designates the database as distributed. This parameter cannot be used if -all is used instead of a *distdbname*.

-all

Specifies that all databases that have not been converted to the current release are to be upgraded.

-f

(Optional) Indicates that one or more Ingres tools catalog product names are specified. A product name must be specified after -f.

Default: No -f flag, which means that all authorized Ingres tools catalogs that are legal for this installation are upgraded (if they exist) or created (if they do not exist). These are the products that the authorization string indicates the installation supports.

The **-f** flag options are as follows:

-f **flag with *products***

Specifies to upgrade (if they exist) or create (if they do not exist) the specified product tools catalogs. Product names are identical to those used with the createdb command.

-f nofeclients

Bypasses upgradefe processing (that is, no tools catalogs are upgraded). This special name cannot be used with any other product name.

product

Specifies one or more product names. Names may include:

- Ingres
- Ingres/dbd
- vision
- windows_4gl
- The special name nofeclients

Examples: Upgradedb

This command upgrades distributed database mystardb and assures that all legal tools catalogs are upgraded (or created if they do not already exist). This is the most common use of upgradedb.

```
upgradedb mystardb
```

or

```
upgradedb mystardb/star
```

This command upgrades distributed database mystardb and bypasses upgrade operations on any tools catalogs that may exist in the database:

```
upgradedb mystardb -f nofeclients
```

or

```
upgradedb mystardb/star -f nofeclients
```

This command upgrades all databases in the installation, including any distributed databases that may reside in the installation. It upgrades only the core product tools catalogs (or creates them if they do not exist):

```
upgradedb -all -f Ingres
```

This command upgrades all databases in the installation, including any distributed databases that may reside in the installation. It upgrades or creates the tools catalogs for all products that this installation is authorized to use:

```
upgradedb -all
```

Ingres Star and Upgradefe

Upgradefe converts or creates tools catalogs for any of the distributed Ingres products. These include Ingres, Ingres/DBD, and Ingres Vision. You may invoke upgradefe directly, using the command syntax described below. Upgradefe is also invoked indirectly by upgradedb.

A Star Server must be running in order to run upgradefe on a distributed database.

Upgradefe cannot be used on coordinator databases.

Upgradefe Command—Update Tools Catalogs

The upgradefe command for distributed databases has the following format:

```
upgradefe distdbname/star {product}
```

distdbname

Specifies the name of a distributed database.

/star

Indicates that the database requires a Star Server because it is a distributed database.

product

Specifies one or more product names. Names may include:

- Ingres
- Ingres/dbd
- vision
- windows_4gl

Examples: Upgradefe

```
upgradefe mystardb/star
```

Assure that the tools catalogs required for the OpenROAD product for distributed database mystardb are upgraded (or created if they do not already exist):

```
upgradefe mystardb/star windows_4gl
```

How You Determine Local and Remote RDBMS Server Releases

To determine the Ingres release at the local and remote nodes, follow these steps:

1. Invoke the Terminal Monitor over the network.

   The release number in the Terminal Monitor's start-up banner shows the release on the local node.

2. Retrieve the value of the _version() system constant.

   The Ingres release at the remote node is shown.

   For more information about system constants, see the *SQL Reference Guide*.

Appendix B: SQL Statement Summary

This appendix lists the SQL statements supported by Ingres Star.

For syntax information, see the *SQL Reference Guide*. For syntax information for Enterprise Access statements, see the *OpenSQL Reference Guide.*

Statements Supported by Ingres Star

Ingres Star supports the following statements.

The tables referred to in these statements are Ingres Star-level objects. Ingres Stars group and role name processing (the -G and -R flags).

Begin Declare Section

The Begin Declare Section begins a program section that declares host variables to embedded SQL. This is a preprocessor directive.

Call

This Call statement calls the operating system or an Ingres subsystem.

Commit

An Ingres Star transaction allows reads to any number of local databases. It allows any updates to any number of local databases providing the local databases updated are managed by non-cluster Ingres DBMSs. If cluster installation Ingres or Enterprise Access products are updating, only one local database can be updated in an Ingres Star transaction.

Ingres Star uses a timeout mechanism to detect any deadlock that may occur between Ingres Star transactions. You must specifically set timeouts to have timeouts of distributed deadlocks.

Connect

Connects the application to a database and, optionally, to a specified distributed transaction.

Copy

The copy statement copies data from a file to a table or vice versa. You may not copy an index or view. Also some Enterprise Access products may not support the copy statement.

Create Link

The create link statement is provided for compatibility with earlier versions of Ingres Star. The register statement is the preferred method for registering local database objects in a distributed database.

Create Table

The create table statement simultaneously creates new tables in a local database and registers them in the distributed database. The table is deleted with the drop statement.

Create View

The create view statement creates a Ingres Star-level distributed view. The view may be defined on other Ingres Star-level objects. Views created with create view are deleted with the drop statement.

Cursor Statements

The Ingres Stars these cursor statements:

- close
- declare
- delete
- fetch
- open
- update

Declare Table

The declare table statement describes the structure of a database table. This is a preprocessor directive.

Delete

The delete statement deletes rows from a table or a view.

Direct Connect

The direct connect statement enables you to connect to an Ingres local DBMS or Enterprise Access directly using Ingres Star in pass-through mode.

Direct Disconnect

The direct disconnect statement enables you to leave the pass-through mode enabled by a previous direct connect statement.

Direct Execute Immediate

The direct execute immediate statement sends a local DBMS-specific statement to a local DBMS.

Disconnect

The disconnect statement terminates access to the database.

Drop

The drop statement removes objects from the distributed database. Only objects created through Ingres Star can be dropped.

Whenever you use the drop statement, you always drop the registration and the underlying locally-stored object. For example, if you use the drop table statement, the underlying table and its data are destroyed. When an Ingres Star-level object is deleted with the drop statement, all views on it also are recursively dropped.

Drop Link

The drop link statement is provided for compatibility with earlier versions of Ingres Star.

Drop Table

The drop table statement removes tables from the distributed database. Any underlying table in a local database also is dropped.

Drop View

The drop view statement removes Ingres Star-level views from the distributed database.

Dynamic SQL

Dynamic SQL compiles and executes SQL queries at run time.

The following statements are supported in the Ingres Star:

Note: You must embed these statements in an application program; you cannot use them interactively.

- declare
- prepare

  Readies a dynamically constructed command string for later execution.
- describe

  Retrieves type, length, and name information about a prepared select statement.
- execute

  Executes a previously prepared dynamic SQL statement.

End Declare Section

The end declare section ends declaration of host variables. This is a preprocessor directive.

Endselect

The Endselect statement terminates a select loop.

Execute Immediate

The execute immediate statement executes an SQL statement specified as a string literal or in a host language variable.

Execute Procedure

The execute procedure statement invokes a database procedure.

Help

A Terminal Monitor statement that displays information about SQL, about the help function itself, or about objects in a database. This statement has these variants:

```
help [[owner.]objectname {, [owner.]objectname}]
help comment column [owner.]table columnname {, columnname}
help comment table [owner.]table {, [owner.]table }
help default [owner.]tablename
help help
help index [owner.]indexname {, [owner.]indexname}
help procedure [owner.]procedure_name
        {, [owner.]procedure_name}
help register [owner.]objectname
help sql [sql_statement]
help table [owner.]tablename {, [owner.]tablename}
help view [owner.]viewname {, [owner.]viewname}
```

Include

The include statement includes an external file in source code.

Inquire_sql

The Inquire_sql statement provides runtime information.

Insert

The insert statement Inserts rows into a table or view in the distributed database.

The column names specified in the statement are the Ingres Star-level column names. The Ingres Star-level column names may be different than the column names in the local DBMS.

Register As Link

The register As link statement registers existing local database tables, views, and database procedures in a distributed database.

Register As Link With Refresh

The Register As Link with Refresh statement updates Ingres Star catalogs when schema or related table information changes in a local table that is part of a distributed database.

Remove

The remove statement removes registrations of tables, views, and procedures from the distributed database previously registered with the register as link statement.

Removes registrations of tables created with the create table statement at the Ingres Star level. When registration of an Ingres Star object is removed with the remove statement, all views on it also are recursively dropped.

Repeat Queries

The repeat queries statement compiles a query for repeated execution during a session.

Note: Repeat queries are accepted syntactically by a Star Server but may not result in enhanced performance. If more than one local database is involved, performance will even be degraded. For this reason, repeat should be used sparingly.

Rollback

The rollback statement rolls back part or all of the current Ingres Star transaction. Rollback to a savepoint is applicable only to an Ingres Star transaction that involves only Ingres local databases.

Savepoint

The savepoint statement declares a savepoint within a transaction.

This statement only applies to transactions that only involve Ingres local databases. Note that some Enterprise Access products do not support the savepoint statement.

Select

The select statement retrieves data from one or more tables, views, or indexes.

Set

The set statement sets an option for the Ingres session, or sets an option for the tables in the distributed database.

If the set option is on a session, the option is sent to each local DBMS currently connected to Ingres Star. Ingres Star also saves the option and sends it to any local DBMS subsequently connected to Ingres Star. Note that the session option cannot be used in a Ingres Star database to alter the lockmode, other than to set the lock timeout. Statements such as "SET LOCKMODE SESSION WHERE READLOCK=NOLOCK" are not supported.

If the set option is on a table, the option is sent to the local DBMS that holds the underlying table.

The set statement has these variants supported by Ingres Star:

- set autocommit on|off

  This statement is not sent to the local DBMS. Ingres Star simply sends a commit statement to each local DBMS when it has completed processing the user's query.
- set ddl_concurrency on|off
- set joinop [no]timeout
- set lockmode
- set [no]optimizeonly
- set [no]printqry
- set [no] qep
- set result_structure
- set update_rowcount changed|qualified

Set_sql

The set_sql statement sets a variety of session characteristics.

Update

The update statement updates the values of columns in a table or a view in the distributed database.

Only table names or view names may be specified with this statement.

Whenever

The whenever statement performs an action when a specified condition becomes true. This is a preprocessor directive.

SQL Statements Not Supported by Ingres Star

Currently, you cannot use the following statements against a distributed database (although most can be executed within a direct connect or a direct execute immediate statement).

The unsupported SQL statements are:

- create|alter|drop location
- declare|alter table
- create|alter|drop user
- Events:
  create dbevent
  drop dbevent
  get dbevent
  raise dbevent
  register dbevent
  remove dbevent
  raise error
- Groups:
  alter group
  create group
  drop group
- comment on
  message
- create index
- declare global temporary table

- Integrities and permits:
  create integrity
  drop integrity
  drop permit
  revoke
- grant
- modify
- Procedures:
  create procedure
  drop procedure
- Roles:
  alter role
  create role
  drop role
- Rules:
  create rule
  drop rule
- Schema:
  create schema
  drop synonym
- Security alarms/audit:
  create security alarm
  drop security_alarm
  enable security_audit
  disable security_audit
- Synonyms:
  create synonym
  drop synonym

The indexes, integrities, and permits of each underlying local DBMS are enforced by the respective local DBMS.

Terminal Monitor Statements Not Supported by Ingres Star

Ingres Star does not support the following terminal monitor statements:

- help constraint
- help dbevent
- help integrity
- help permit
- help rule

Inside a direct connect, these statements provide help on permits and integrities in the local database. Outside of a direct connect, these statements return an error message that no permits or integrities exist.

Appendix C: Standard Catalog Interface

This appendix describes the formats of the Standard Catalog Interface for Ingres Star.

Standard Catalog Interface and Ingres Star

The standard catalog interface lets you get information from the Ingres Star system catalogs, which cannot be queried directly. This interface can be used in programs to access (but not update) information about the system.

Catalog Formats

Database users may read the standard catalogs, but may not update them.

- To display them, use the interactive query language help table or help view statement, as appropriate, or use Visual DBA.
- Unless otherwise noted, values in system catalogs are left justified, and columns are non-nullable.
- The length of char fields, as listed in the Data Type column, is a maximum length; the actual length of the field is installation-dependent. When developing applications that access these catalogs, you should allocate storage on the basis of the length as shown in the Data Type column.
- All dates stored in system catalogs have the following format, with underscores and colons required:

  yyyy_mm_dd hh:mm:ss GMT

  where:

  yyyy is the year (for example, 2000)

  mm is the month (for example, 11)

  dd is the day of the month (for example, 21)

  hh is the military hour (for example, 13)

  mm is the minute (for example, 43)

  ss is the second (for example, 32)

  GMT is Greenwich Mean Time

Catalogs

The formats of the Standard Catalog Interface for Ingres Star are described in detail in tables in the following sections. All database users can read the Standard Catalog Interface, but users should never update the catalogs.

The iialt_columns Catalog

For each alternate key, any columns defined as part of the key have an entry in iialt_columns.

Column Name	Data Type	Description
table_name	char(32)	The table to which column_name belongs
table_owner	char(32)	The table owner
key_id	integer	The number of the alternate key for this table
column_name	char(32)	The name of the column
key_sequence	smallint	Sequence of column within the key, numbered from 1

The iicolumns Catalog

For each querytable object in the iitables catalog, there are one or more entries in the iicolumns catalog. Each row in iicolumns contains the logical information on a column of the query object. The iicolumns catalog is used by user interfaces and user programs to perform dictionary operations and dynamic queries.

Column Name	Data Type	Description
table_name	char(32)	The name of the table. Must be a valid name.
table_owner	char(32)	The owner of the table. Must be a valid username.
column_name	char(32)	The column's name. Must be a valid name.

Column Name	Data Type	Description
column_datatype	char(32)	The column's data type name returned to users and applications: • integer • smallint • int • float • real • double precision • char • character • varchar • c • text • date • money • decimal • user data types (UDTs) For details on UDTs, see the *Object Management Extension User Guide*.
column_length	integer	The length of the column returned to users and applications. If a data type contains two length specifiers, this column uses the first length. Set to zero for the data types which are specified without length (money and date). This length is not the actual length of the column's internal storage.
column_scale	smallint	The second number in a two-part user length specification; for typename (len1, len2) it will be len2.
column_nulls	char(8)	Y if the column can contain null values, N if not.
column_defaults	char(8)	Y if the column is given a default value when a row is inserted. N if not.

Column Name	Data Type	Description
column_sequence	smallint	The number of this column in the corresponding table's create statement, numbered from 1.
key_sequence	smallint	The order of this column in the primary key, numbered from 1. For a table, this indicates the column's order in the primary storage structure key. If 0, then this column is not part of the primary key. This is unique if the unique_rule column for the table's corresponding entry in iitables is set to U.
Sort_direction	char(8)	Defaults to A for ascending when key_sequence is greater than 0. Otherwise, this value is a blank.
Column_ingdatatype	smallint	Contains a value that indicates the data type of the column. If the value is positive then the column is not nullable; if the value is negative, then the column is nullable. The data types and their corresponding values are: □ integer -30/30 □ float -31/31 □ c -32/32 □ text- 37/37 □ date -3/3 □ money -5/5 □ char -20/20 □ varchar -21/21 □ decimal -10/10

The iidbcapabilities Catalog

The iidbcapabilities catalog contains information about the capabilities provided by the DBMS. The following table describes the columns in the iidbcapabilities catalog:

Column Name	Data Type	Description
cap_capability	char(32)	Contains one of the values listed in the Capability column of the following table. If the cap_capability has a value, it will be activated by the value in the cap_value column.
Cap_value	char(32)	Set to the value of the capability. This is usually the string Y or N. See the Values column in the following table for possible values of each capability.

The cap_capability column in the iidbcapabilities catalog contains one or more of the following values:

Capability	Value
OPEN/SQL_LEVEL	Version of OpenSQL supported by the DBMS. Use this name in preference to the older COMMON/SQL_LEVEL. Default setting is 00605.
COMMON/SQL_LEVEL	Version of OPEN/SQL support provided by the DBMS. Maintained for backward compatibility. (Use OPEN/SQL_LEVEL instead.) Examples: • 00600 Version 6.0 • 00601 Version 6.1 • 00602 Version 6.2 Default is 00600.
DB_NAME_CASE	Case mapping semantics of the database with respect to regular identifiers for database objects: LOWER for lowercase is the Ingres setting. UPPER for uppercase is set for an ISO Entry SQL92 compliant installation.

Capability	Value
DB_DELIMITED_CASE	Case mapping semantics of the database with respect to delimited identifiers for database objects: LOWER for lowercase is the Ingres setting. MIXED for mixed case is set for an ISO Entry SQL92 compliant installation. If the value is MIXED, an identifier must be enclosed in double quotes to maintain case as originally defined. Otherwise it is treated as a regular identifier (converted to uppercase).
DB_REAL_USER_CASE	Case mapping of user names as retrieved by the operating system: LOWER for lowercase is the Ingres setting. MIXED for mixed case or UPPER for uppercase is set as specified during installation.
DBMS_TYPE	The type of DBMS the application is communicating with. For a Star Server, the value is always STAR.
DISTRIBUTED	Y since the DBMS is distributed.
INGRES	Set to Y if the DBMS supports in all respects 100 percent of Ingres Release 6. Otherwise N. For Ingres Star this is set to N since it does not support QUEL.
INGRES/SQL_LEVEL	Version of SQL supported by the DBMS. These include: □ 00600 Version 6.0 □ 00601 Version 6.1 □ 00602 Version 6.2 □ 00603 Version 6.3 □ 00604 Version 6.4 □ 00605 Ingres □ 00000 DBMS does not support SQL Default is 00600.

Capability	Value
INGRES/QUEL_LEVEL	Version of QUEL supported by the DBMS. These include: • 00600 Version 6.0 • 00601 Version 6.1 • 00602 Version 6.2 • 00603 Version 6.3 • 00604 Version 6.4 • 00605 Ingres • 00000 DBMS does not support QUEL Default is 00600.
OWNER_NAME	Contains N if schema.table table name format is not supported. Contains Y if schema.table format is supported; contains QUOTED if schema.table is supported with optional quotes ("schema".table). The default is QUOTED.
PHYSICAL_SOURCE	T indicates that both iitables and iiphysical_tables contain physical table information. P indicates that only iiphysical_tables contains the physical table information.
SAVEPOINTS	Y if savepoints behave exactly as in Ingres, else N. Default is Y.
STANDARD_CATALOG _LEVEL	Version of the standard catalog interface supported by this database. Should be 00602 (the default) for Ingres Star.
UNIQUE_KEY_REQ	Set to Y if the database service requires that some or all tables have a unique key. Set to N or not present if the database service allows tables without unique keys.

The iidbconstants Catalog

The iidbconstants catalog contains values required by the Ingres tools. The following table describes the columns in the iidbconstants catalog:

Column Name	Data Type	Description
user_name	varchar(32)	The name of the current user.
dba_name	varchar(32)	The name of the db's owner.
system_owner	varchar(32)	The name of the catalog owner (for example, $ingres).

The iihistograms Catalog

The iihistograms table contains histogram information used by the optimizer:

Column Name	Data Type	Description
table_name	char(32)	The table for the histogram. Must be a valid name.
table_owner	char(32)	The table owner. Must be a valid name.
column_name	char(32)	The name of the column. Must be a valid name.
text_sequence	Integer	The sequence number for the histogram, numbered from 1. There may be several rows in this table, used to order the optdata data when histogram is read into memory.
text_segment	char(228)	The encoded histogram data, created by optimizedb.

The iiindex_columns Catalog

For indexes, any columns that are defined as part of the primary index key will have an entry in iiindex_columns. For a full list of all columns in the index, use the iicolumns catalog.

Column Name	Data Type	Description
index_name	char(32)	The index containing *column_name.* Must be a valid name.

Column Name	Data Type	Description
index_owner	char(32)	The index owner. Must be a valid username.
column_name	char(32)	The name of the column. Must be a valid name.
key_sequence	smallint	Sequence of column within the key, numbered from 1.
sort_direction	char(8)	Defaults to A for ascending.

The iiindexes Catalog

Each table with a table_type of I in the iitables table has an entry in iiindexes. All indexes also have an entry in iiphysical_tables.

Column Name	Data Type	Description
index_name	char(32)	The index name. Must be a valid name.
index_owner	char(32)	The index owner. Must be a valid username.
create_date	char(25)	Creation date of index. This is a date field.
base_name	char(32)	The base table name. Must be a valid name.
base_owner	char(32)	The base table owner. Must be a valid username.
storage_structure	char(16)	The storage structure for the index: heap, hash, isam, or B-tree. Set to blank if unknown.
is_compressed	char(8)	Y if the table is stored in compressed format, N if the table is uncompressed, blank if unknown.
unique_rule	char(8)	U if the index is unique, D if duplicate key values are allowed, or blank if unknown.

The iiintegrities Catalog

Iiintegrities contains one or more entries for each integrity defined on a table. Because the text of the integrity definition can contain more than 240 characters, iiintegrities may contain more than one row for a single integrity. The text may contain newlines and may be broken mid-word across rows.

Note: Ingres Star does not support integrities, so there are no rows in this catalog.

This table is keyed on table_name and table_owner:

Column Name	Data Type	Description
table_name	char(32)	The table name. Must be a valid name.
table_owner	char(32)	The table owner. Must be a valid name.
create_date	char(25)	The integrity's creation date. This is a date field.
integrity_number	smallint	The number of this integrity.
text_sequence	smallint	The sequence number for the text, numbered from 1.
text_segment	varchar (240)	The text of the integrity definition.

The iimulti_locations Catalog

For tables located on multiple volumes, this table contains an entry for each additional location on which a table resides. The first location for a table can be found in the iitables catalog.

Note: Ingres Star does not currently populate this table.

This table is keyed on table_name and table_owner:

Column Name	Data Type	Description
table_name	char(32)	The table name. Must be a valid name.
table_owner	char(32)	The table's owner. Must be a valid username.
sequence	integer	The sequence of this location in the list of locations, as specified in the modify statement. This is numbered from 1.

Column Name	Data Type	Description
location_name	char(32)	The name of the location.

The iipermits Catalog

The iipermits catalog contains one or more entries for each permit defined. Because the permit definition can contain more than 240 characters, iipermits can contain more than one row for a single permit. The text may contain newlines and may be broken mid-word across rows.

Note: Ingres Star does not currently support permits, so there are no rows in this catalog.

This table is keyed on object_name and object_owner:

Column Name	Data Type	Description
object_name	char(32)	The table or procedure name. Must be a valid name.
object_owner	char(32)	The owner of the table or procedure. Must be a valid name.
object_type	char(8)	The type of the object: T for a table or view; P for a database procedure.
create_date	char(25)	The permit's creation date. This is a date field.
permit_user	char(32)	The username to which this permit applies.
permit_number	smallint	The number of this permit.
text_sequence	smallint	The sequence number for the text, numbered from 1.
text-segment	varchar (240)	The text of the permission definition.

The iiphysical_tables Catalog

The information in the iiphysical_tables catalog overlaps with some of the information in iitables. You can query the physical_source column in iidbcapabilities to determine whether you must query iiphysical_tables. If you do not want to make this check, then you must always query iiphysical_tables to be sure of getting the correct information.

If a queryable object is type T (table), then it is a physical table and may have an entry in iiphysical_tables as well as iitables.

Column Name	Data Type	Description
table_name	char(32)	The table name. This is an name.
table_owner	char(32)	The table owner's username.
table_stats	char(8)	Y if this object has entries in the iistats table, N if it does not. If blank, it is undetermined if the object has entries in iistats and you should check iistats directly.
table_indexes	char(8)	Y if this object has entries in the iiindexes table that refer to this as a base table, N if not. If blank, it is undetermined if the object has entries in the iiindexes table that refer to it as a base table, and you should check iiindexes directly. This field is only used for optimization for Ingres databases, as other Enterprise Access products cannot automatically supply this information.
is_readonly	char(8)	Y if updates are physically allowed on this object, N if not. The field is blank if this is unknown. This is used for tables that are defined to the Enterprise Access for retrieval. If this field is set to Y, updates will not be allowed regardless of what permissions might be set for the table.
num_rows	integer	The estimated number of rows in the table. Set to -1 if unknown.
storage_structure	char(16)	The storage structure of the table. Possible values are: heap, B-tree, isam, or hash. Set to blank if the structure is unknown.

Column Name	Data Type	Description
is_compressed	char(8)	Indicates if the table is stored in compressed format. Y if it is compressed, N if not compressed, blank if unknown.
duplicate_rows	char(8)	Indicates if duplicate rows are allowed in the table. Set to U if rows must be unique, D if duplicates are allowed, or blank if unknown.
unique_rule	char(8)	Indicates if the storage structure key is unique. Set to U if the storage structure is unique, D if duplicates are allowed, blank if unknown or inapplicable.
number_pages	integer	The estimated number of physical pages in the table. Set to -1 if unknown.
overflow_pages	integer	The estimated number of overflow pages in the table. Set to -1 if unknown.
row_width	integer	The size (in bytes) of the uncompressed binary value for a row in the object for Ingres. Set to -1 if this is unknown.

The iiprocedures Catalog

The iiprocedures catalog contains one or more entries for each procedure defined. Because the procedure definition can contain more than 240 characters, iiprocedures can contain more than one row for a single procedure. The text may contain newlines and may be broken mid-word across rows. The text segment contains the procedure registration text, not the actual procedure definition text.

This table is keyed on procedure_name and procedure_owner:

Column Name	Data Type	Description
procedure_name	char(32)	The table or procedure name. Must be a valid name.
procedure_owner	char(32)	The owner of the table or procedure. Must be a valid name.
create_date	char(25)	The permit's creation date. This is a date field.

Column Name	Data Type	Description
proc_subtype	varchar(1)	The type of procedure, which is one of the following values: • N (native) = the database supports standard Ingres database procedures • I (import) = for Enterprise Access products, the database supports host DBMS procedures • E (external) = for Enterprise Access products, the database supports procedures external to the database • L (link) = a Ingres Star registered procedure
text_sequence	integer	The sequence number for the text, numbered from 1.
text-segment	varchar (240)	The text of the procedure definition.

The iiregistered_ objects Catalog

The iiregistered_objects catalog is an Ingres Star-only catalog. It resides in all distributed databases, but is not available in local databases. This catalog ties registered objects that Ingres Star may acquire when the user registers a table to the underlying objects in the local database. The registered objects are tables, views, secondary indexes, and procedures.

Column Name	Data Type	Description
ddb_object_name	char(32)	The name of the Ingres Star-registered object. Must be a valid name.
ddb_object_owner	char(32)	The name of the owner of the Ingres Star-registered object. Must be a valid username.
register_date	char(25)	The date the object was registered. This is a date field.
ldb_database	char(32)	The name of the local database in which the registered object resides.
ldb_node	char(32)	The node on which the ldb_database resides.
ldb_dbmstype	char(32)	The type of the ldb_database. These are the same types used by iinamu (INGRES, RMS, DB2, RDB, and so on).

Column Name	Data Type	Description
ldb_object_name	char(32)	The name that the local database uses for the registered object.
ldb_object_owner	char(32)	The name of the owner of the registered object in the local database.
ldb_object_type	char(8)	The type of local object. The values are T if the object is a table, V if it is a view, I if the object is an index, or P if the object is a procedure.

The iiregistrations Catalog

The iiregistrations catalog contains the text of register statements.

Column Name	Data Type	Description
object_name	char(32)	The name of the registered table, view, or index.
object_owner	char(32)	The name of the owner of the table, view, or index.
object_dml	char(8)	The language used in the registration statement. S for SQL.
object_type	char(8)	Describes the object type of object_name. The values are T if the object is a table, V if it is a view, I if the object is an index, or P if the object is a registered procedure.
object_subtype	char(8)	Describes the type of table or view created by the register statement. For Ingres Star, this will be L for a link.
text_sequence	integer	The sequence number of the text field, numbered from 1.
Text_segment	varchar(240)	The text of the register statement.

The iistats Catalog

The iistats catalog contains the following information.

Column Name	Data Type	Description
table_name	char(32)	The name of the table. Must be a valid name.
table_owner	char(32)	The table owner. Must be a valid username.
column_name	char(32)	The column name to which the statistics apply. Must be a valid name.
create_date	char(25)	The date on which statistics were gathered. This is a date field.
num_unique	float4	The number of unique values in the column.
rept_factor	float4	The repetition factor, or the inverse of the number of unique values (number of rows/ number of unique values).
has_unique	char(8)	Y if the column has unique values, N otherwise.
pct_nulls	float4	The percentage (fraction of 1.0) of the table which contains NULL for the column.
num_cells	smallint	The number of cells in the histogram.
column_domain	smallint	A user-specified number signifying the domain from which the column draws its values; default is 0.
is_complete	char(8)	Y if the column contains all possible values in the domain, N otherwise.
stat_version	char(8)	The version of the statistics for this column.
hist_data_length	smallint	The length of the histogram boundary values, either the user-specified length or optimizedb's computed length.

The iitables Catalog

The iitables catalog contains an entry for each queryable object in the database (table, view, or index). To find out which tables, views, and indexes are owned by you or the DBA, you can query this catalog; for example:

```
select * from iitables
  where (table_owner = user
    or table_owner = dba())
```

Column Name	Data Type	Description
table_name	char(32)	The object's name. Must be a valid name.
table_owner	char(32)	The object's owner. Must be a valid username. Generally, the creator of the object is the owner.
create_date	char(25)	The object's creation date. Blank if unknown. This is a date field.
alter_date	char(25)	The last time this table was altered. This date is updated whenever the logical structure of the table changes, either through changes to the columns in the table or changes to the primary key. Physical changes to the table, such as changes to data, secondary indexes, or physical keys, do not change this date. Blank if unknown. This is a date field.
table_type	char(8)	Type of the query object: □ T table □ V view □ I index Further information about tables can be found in iiphysical_tables; further information about views can be found in iiviews.
table_subtype	char(8)	Specifies the type of table or view. Possible values are: □ N (native) for Ingres Star-level table (created by create table or create view statement issued from Ingres Star) □ L (links) for Ingres Star □ " " (blank) if unknown

Column Name	Data Type	Description
table_version	char(8)	Version of the object; enables the tools to determine where additional information about this particular object is stored. This reflects the database type, as well as the version of an object within a given database. For tables, the value for this field is ING6.0.
system_use	char(8)	Contains S if the object is a system object, U if user object, or blank if unknown. Used by utilities to determine which tables need reloading. If the value is unknown, the utilities will use the naming convention of ii for tables in order to distinguish between system and user catalogs. Also, any table beginning with ii_ is assumed to be a tool object, rather than a DBMS system object. The system catalogs themselves must be included in the iitables catalog and are considered system tables.

The following information may also be present in iiphysical_tables but not present in this catalog:

Column Name	Data Type	Description
table_stats	char(8)	Y if this object has entries in the iistats table, N if this object does not have entries. If this field is blank, then you must query iistats to determine if statistics exist. This column is used only for optimization of databases.
table_indexes	char(8)	Y if this object has entries in the iiindexes table that refer to this as a base table, or N if this object does not have entries. If the field is blank, then you must query iiindexes on the base_table column. This field is used only for optimization of databases.
is_readonly	char(8)	N if updates are physically allowed, Y if no updates are allowed, or blank if unknown. Used for tables that are defined to the Enterprise Access only for retrieval, such as tables in hierarchical database systems. If this field is set to Y then no updates will work, independent of what permissions might be set. If it is set to N, updates may be allowed, depending on whether the permissions allow it or not.

Column Name	Data Type	Description
num_rows	integer	The estimated number of rows in the table. Set to -1 if unknown.
storage_structure	char(16)	The storage structure for the table: heap, hash, B-tree, or isam. Blank if the table structure is unknown.
is_compressed	char(8)	Y if the table is stored in compressed format, N if the table is uncompressed, blank if unknown.
duplicate_rows	char(8)	D if the table allows duplicate rows, U if the table does not allow duplicate rows, blank if unknown. The table storage structure (unique vs. non-unique keys) can override this setting.
unique_rule	char(8)	The value may be U (unique key), D (duplicate key) or blank if unknown or does not apply. D indicates that duplicate physical storage structure keys are allowed. (A unique alternate key may exist in iialt_columns and any storage structure keys may be listed in iicolumns.) U: If the object is an Ingres object, indicates that the object has a unique storage structure key(s); if the object is not an Ingres object, then it indicates that the object has a unique key, described in either iicolumns or iialt_columns.
number_pages	integer	The estimated number of physical pages in the table. Set to -1 if unknown.
overflow_pages	integer	The estimated number of overflow pages in the table. Set to -1 if unknown.
row_width	integer	The size, in bytes, of the uncompressed binary value for a row of this query object.

The information in the following table is not duplicated in iiphysical_tables:

Column Name	Data Type	Description
expire_date	integer	Expiration date of table. This is a _bintime date.

Column Name	Data Type	Description
modify_date	char(25)	The date on which the last physical modification to the storage structure of the table occurred. Blank if unknown or inapplicable. This is a date field.
location_name	char(32)	The first location of the table. If there are additional locations for a table, they will be shown in the iimulti_locations table and multi_locations will be set to Y.
table_integrities	char(8)	Y if this object has Ingres-style integrities. If the value is blank, you must query the iiintegrities table to determine if integrities exist.
table_permits	char(8)	Y if this object has Ingres-style permissions.
all_to_all	char(8)	Y if this object has permit all to all, N if not.
ret_to_all	char(8)	Y if this object has permit retrieve to all, N if not.
is_journalled	char(8)	Y if journaling is enabled on this object, N if not. Set to C if journaling will be enabled at the next checkpoint. This will be blank if journaling does not apply.
view_base	char(8)	Y if object is a base for a view definition, N if not, or blank if unknown.
multi_locations	char(8)	Y if the table is in multiple locations, N if not.
table_ifillpct	smallint	Fill factor for the index pages used on the last modify statement in the nonleaffill clause, expressed as a percentage (0 to 100). Used for B-tree structures in order to rerun the last modify statement.
table_dfillpct	smallint	Fill factor for the data pages used on the last modify statement in the fillfactor clause, expressed as a percentage (0 to 100). Used for B-tree, hash, and isam structures in order to rerun the last modify statement.
table_lfillpct	smallint	Fill factor for the leaf pages used on the last modify statement in the leaffill clause, expressed as a percentage (0 to 100). Used for B-tree structures in order to rerun the last modify statement.

Column Name	Data Type	Description
table_minpages	integer	Minpages parameter from the last execution of the modify statement. Used for hash structures only.
table_maxpages	integer	Maxpages parameter from the last execution of the modify statement. Used for hash structures only.
table_relstamp1	integer	High part of last create or modify timestamp for the table.
table_relstamp2	integer	Low part of last create or modify timestamp for the table.
table_reltid	integer	The first part of the internal relation ID.
table_reltidx	integer	The second part of the internal relation ID.

The iiviews Catalog

The iiviews catalog contains one or more entries for each view in the database. (Views are indicated in iitables by table type = V.) Because the text_segment column is limited to 256 characters per row, a single view can require more than one row to contain all its text; in this case, the text will be broken in mid-word across the sequenced rows. The text column is text and may contain newline characters.

Column Name	Data Type	Description
table_name	char(32)	The view name. Must be a valid Ingres name.
table_owner	char(32)	The view owner's Ingres username.
view_dml	char(8)	The language in which the view was created: S (for SQL).
check_option	char(8)	Y if the check option was specified in the create view statement, or N if not. Set to blank if unknown.
text_sequence	integer	The sequence number for the text field, starting with 1.
text_segment	varchar(256)	The text of the view definition.

Index

E

F

G

H

I

L

N

O

P

Q

R

S

T

U

V

W