InterBase 6

Operations Guide

Borland®

SOFTWARE CORPORATION

100 Enterprise Way, Scotts Valley, CA 95066 http://www.borland.com

INT0060WW21000 20000

D4

Table of Contents

CHAPTER 1 **Introduction**

CHAPTER 2 **IBConsole: The InterBase Interface**

CHAPTER 3 **Server Configuration**

CHAPTER 4 **Network Configuration**

CHAPTER 5 **Database Security**

CHAPTER 6 **Database Configuration and Maintenance**

CHAPTER 7 **Database Backup and Restore**

CHAPTER 8 **Database and Server Statistics**

CHAPTER 9 **Interactive Query**

List of Tables

List of Figures

CHAPTER 1 Introduction

The InterBase *Operations Guide* is a task-oriented reference of procedures to install, configure, and maintain an InterBase database server or Local InterBase workstation.

This chapter describes who should read this book, and provides a brief overview of the capabilities and tools available in the InterBase product line.

Who should use this guide

The InterBase *Operations Guide* is for database administrators or system administrators who are responsible for operating and maintaining InterBase database servers. The material is also useful for application developers who wish to understand more about InterBase technology. The guide assumes knowledge of:

- Server operating systems including Windows NT/2000, Linux, and UNIX
- Networks and network protocols
- Application programming

Topics covered in this guide

- Introduction to InterBase features
- Server configuration, startup and shutdown
- Network configuration and troubleshooting guidelines
- Security configuration for InterBase servers, databases, and data; reference for the security configuration tools
- Database configuration and maintenance options; reference for the maintenance tools
- Backing up and restoring databases; reference for the backup tools
- Performance troubleshooting and tuning guidelines.
- Database and server statistics monitoring
- Interactive query profiling; reference for the interactive query tools

System requirements and server sizing

InterBase server runs on a variety of platforms, including Microsoft Windows NT, Windows 2000, and Windows 98/ME, Linux, and several UNIX operating systems.

The InterBase server software makes efficient use of system resources on the server node. The server process uses little more than 1.9MB of memory. Typically, each client connection to the server adds approximately 115KB of memory. This varies based on the nature of the client applications and the database design, so the figure is only a baseline for comparison.

The minimal software installation requires disk space ranging from 9MB to 12MB, depending on platform. During operation, InterBase's sorting routine requires additional disk space as scratch space. The amount of space depends on the volume and type of data the server is requested to sort.

The InterBase client also runs on any of these operating systems. In addition, InterBase provides the InterClient Java client interface using the JDBC standard for database connectivity. Client applications written in Java can run on any client platform that supports Java, even if InterBase does not explicitly list it among its supported platforms. Examples include the Macintosh and Internet appliances with embedded Java capabilities.

For more information on system requirements, including a complete listing of supported hardware, software, and system configurations, see the System Requirements chapter in the *Getting Started* manual. Information about system requirements can also be found at http://www.borland.com.

Primary InterBase features

InterBase offers all the benefits of a full-featured RDBMS. The following table lists some of the key InterBase features:

Feature	Description
Network protocol support	• All platforms of InterBase support TCP/IP • InterBase server for Windows NT/2000 and all Windows clients support NetBEUI/named pipes • InterBase Server for NetWare and all Windows clients support IPX/SPX
SQL-92 entry-level conformance	ANSI standard SQL, available through an Interactive SQL tool and Borland desktop applications
Simultaneous access to multiple databases	One application can access many databases at the same time
multigenerational architecture	Server maintains older versions of records (as needed) so that transactions can see a consistent view of data
Optimistic row-level locking	Server locks only the individual records that a client updates, instead of locking an entire database page
Query optimization	Server optimizes queries automatically, or you can manually specify a query plan
Blob datatype and Blob filters	Dynamically sizeable datatypes that can contain unformatted data such as graphics and text
Declarative referential integrity	Automatic enforcement of cross-table relationships (between FOREIGN and PRIMARY KEYs)
Stored procedures	Programmatic elements in the database for advanced queries and data manipulation actions
Triggers	Self-contained program modules that are activated when data in a specific table is inserted, updated, or deleted

TABLE 1.1 InterBase features

Feature	**Description**
Event alerters	Messages passed from the database to an application; enables applications to receive asynchronous notification of database changes
Updatable views	Views can reflect data changes as they occur
User-defined functions (UDFs)	Program modules that run on the server
Outer joins	Relational construct between two tables that enables complex operations
Explicit transaction management	Full control of transaction start, commit, and rollback, including named transactions
Concurrent multiple application access to data	One client reading a table does not block others from it
multidimensional arrays	Column datatypes arranged in an indexed list of elements
Automatic two-phase commit	Multi-database transactions check that changes to all databases happen before committing (InterBase Server only)
InterBase API	Functions that enable applications to construct SQL/DSQL statements directly to the InterBase engine and receive results back
gpre	Preprocessor for converting embedded SQL/DSQL statements and variables into a format that can be read by a host-language compiler
IBConsole	Windows tool for data definition, query, database backup, restoration, maintenance, and security
isql	Command-line version of the InterBase interactive SQL tool; can be used instead of IBConsole.

TABLE 1.1 InterBase features (*continued*)

Feature	Description
Command-line database administrator utilities	Command-line version of the InterBase database administration tools; can be used instead of IBConsole
Header files	Files included at the beginning of application programs that define InterBase datatypes and function calls
Example make files	Files that demonstrate how to invoke the makefiles to compile and link InterBase applications
Example programs	C programs, ready to compile and link, which you can use to query standard InterBase example databases on the server
Message file	**interbase.msg**, containing messages presented to the user

TABLE 1.1 InterBase features (*continued*)

SQL support

InterBase conforms to entry-level SQL-92 requirements. It supports declarative referential integrity with cascading operations, updatable views, and outer joins. InterBase Server provides libraries that support development of embedded SQL and DSQL client applications. On all InterBase platforms, client applications can be written to the InterBase API, a library of functions with which to send requests for database operations to the server.

InterBase also supports extended SQL features, some of which anticipate SQL99 extensions to the SQL standard. These include stored procedures, triggers, SQL roles, and segmented Blob support.

For information on SQL, see the *Language Reference*.

Multiuser database access

InterBase enables many client applications to access a single database simultaneously. A client applications can also access the multiple databases simultaneously. SQL triggers can notify client applications when specific database events occur, such as insertions or deletions.

You can write user-defined functions (UDFs) and store them in an InterBase database, where they are accessible to all client applications accessing the database.

Transaction management

Client applications can start multiple simultaneous transactions. InterBase provides full and explicit transaction control for starting, committing, and rolling back transactions. The statements and functions that control starting a transaction also control transaction behavior.

InterBase transactions can be isolated from changes made by other concurrent transactions. For the life of these transactions, the database appears to be unchanged except for the changes made by the transaction. Records deleted by another transaction exist, newly stored records do not appear to exist, and updated records remain in the original state.

For information on transaction management, see the *Embedded SQL Guide*.

Multigenerational architecture

InterBase provides expedient handling of time-critical transactions through support of data concurrency and consistency in mixed use—query and update—environments. InterBase uses a multigenerational architecture, which creates and stores multiple versions of each data record. By creating a new version of a record, InterBase allows all clients to read a version of any record at any time, even if another user is changing that record. InterBase also uses transactions to isolate groups of database changes from other changes.

Optimistic row-level locking

Optimistic locks are applied only when a client actually updates data, instead of at the beginning of a transaction. InterBase uses optimistic locking technology to provide greater throughput of database operations for clients.

InterBase implements true row-level locks, to restrict changes only to the records of the database that a client changes; this is distinct from page-level locks, which restrict any arbitrary data that is stored physically nearby in the database. Row-level locks permit multiple clients to update data that is in the same table without coming into conflict. This results in greater throughput and less serialization of database operations.

InterBase also provides options for pessimistic table-level locking. See the *Embedded SQL Guide* for details.

Database administration

InterBase provides both GUI and command-line tools for managing databases and servers. You can perform database administration on databases residing on Local InterBase or InterBase Server with IBConsole, a Windows application running on a client PC. You can also use command-line database administration utilities on the server.

IBConsole and command-line tools enable the database administrator to:

- Manage server security
- Back up and restore a database
- Perform database maintenance
- View database and lock manager statistics

You can find more information on server security later in this chapter, and later chapters describe individual tasks you can accomplish with IBConsole and the command-line tools.

Managing server security

InterBase maintains a list of user names and passwords in a security database. The security database allows clients to connect to an InterBase database on a server if a user name and password supplied by the client match a valid user name and password combination in the security database, **isc4.gdb**, on the server.

You can add and delete user names and modify a user's parameters, such as password and user ID.

For information about managing server security, see **Chapter 5, "Database Security."**

Backing up and restoring databases

You can backup and restore a database using IBConsole or command-line **gbak**. A backup can run concurrently with other processes accessing the database because it does not require exclusive access to the database.

Database backup and restoration can also be used for:

- Erasing obsolete versions of database records
- Changing the database page size
- Changing the database from single-file to multifile
- Transferring a database from one operating system to another
- Backing up only a database's metadata to recreate an empty database

For information about database backup and recovery, see **Chapter 7, "Database Backup and Restore."**

Maintaining a database

You can prepare a database for shutdown and perform database maintenance using either IBConsole or the command-line utilities. If a database incurs minor problems, such as an operating system write error, these tools enable you to sweep a database without taking the database off-line.

Some of the tasks that are part of database maintenance are:

- Sweeping a database
- Shutting down the database to provide exclusive access to it
- Validating table fragments
- Preparing a corrupt database for backup
- Resolving transactions "in limbo" from a two-phase commit
- Validating and repairing the database structure

For information about database maintenance, see **Chapter 6, "Database Configuration and Maintenance."**

Viewing statistics

You can monitor the status of a database by viewing statistics from the database header page, and an analysis of tables and indexes. For more information, see **Chapter 8, "Database and Server Statistics."**

About InterBase SuperServer architecture

InterBase 6 uses SuperServer architecture: a multi-client, multi-threaded implementation of the InterBase server process. SuperServer replaces the Classic implementation used for previous versions of InterBase. SuperServer serves many clients at the same time, using threads instead of separate server processes for each client. Multiple threads share access to a single server process.

How SuperServer differs from Classic architecture

If you are upgrading from a previous version of InterBase, it will help to understand how the differences listed in the table below affect database operations. The differences are explained in the paragraphs that follow the table.

Feature	Classic	SuperServer
Executable	**gds_inet_server**	ibserver
Processes	Multiple, on demand, one instance per client	single, threaded, one thread per client
Lock management	**gds_lock_mgr** process	Implemented as a thread
Resource use	One cache per process	One cache space for all clients
Local access	Direct I/O	I/O by proxy (network style)
Security on UNIX	Executable and Lock Manager	Can run as non-root **uid**, with access restrictions

TABLE 1.2 Comparison of Classic and SuperServer architectures

▸ *Executable and processes*

- **Classic**: Runs on demand as multiple processes. When a client attempts to connect to an InterBase database, one instance of the **gds_inet_server** executable runs and remains dedicated to that client connection for the duration of the connection.
- **SuperServer**: Runs as a single process, an invocation of the **ibserver** executable. **ibserver** is started once by the system administrator or by a system boot script. This process runs always, waiting for connection requests. Even when no client is connected to a database on the server, **ibserver** continues to run quietly. The SuperServer process does not depend on **inetd**; it waits for connection requests to the **gds_db** service itself.

▸ *Lock management*

- **Classic**: For every client connection a separate server process is started to execute the database engine, and each server process has a dedicated database cache. The server processes contend for access to the database, so a Lock Manager subsystem is required to arbitrate and synchronize concurrent page access among the processes.
- **SuperServer**: The lock manager is implemented as a thread in the **ibserver** executable. Therefore, InterBase does not use the **gds_lock_mgr** process. Likewise, POSIX signals are not used by the lock manager thread in SuperServer; interthread communication mechanisms are used instead.

Resource use

- **Classic**: Each instance of **gds_inet_server** keeps a cache of database pages in its memory space. While the resource use per client is greater than in SuperServer, Classic uses less overall resources when the number of concurrent connections is low.
- **SuperServer**: One cache space is used for all client attachments, allowing more efficient use of cache memory.

Local access method

- **Classic**: Permits application processes that are running on the same host as the database to perform I/O on database files directly.
- **SuperServer**: Requires applications to request **ibserver** I/O operations by proxy, using a network method. SuperSaver currently does not support true local access as Classic does, even when the client application and **ibserver** are running on the same host. This is important, because Classic's use of direct I/O is faster than the interprocess communication method that is required when using SuperServer.

Security on UNIX platforms

- **Classic**: Server executables **gds_inet_server** and **gds_lock_mgr** must run as root. The lock manager must have the superuser privilege to send signals to the processes.
- **SuperServer**: Can be configured to run as a non-root **uid**, for enhanced security. The permissions on database files can be restricted to allow only InterBase server **uid** to access the database.

Which is better?

Given as abstract workload, neither architecture is a clear winner. Given a specific workload, one architecture generally outshines the other:

- A single application running on the server machine is faster in Classic mode.
- For an application embedded in an appliance, Classic is better, because it provides a single process from user to disk.
- On a mono-processor, an application with several dozen highly contentious clients is faster with SuperServer, because of the shared cache.
- On dual processors, two clients that do not interact are much better in Classic mode.

Overview of command-line tools

For each task that you can perform in IBConsole, there is a command-line tool that you can run in a command window or console to perform the same task.

The UNIX versions of InterBase include all of the following command-line tools. The graphical Windows tools do not run on a UNIX workstation, though you can run most of the tools on Windows to connect to and operate on InterBase databases that reside on UNIX servers.

An advantage of noninteractive, command-line tools is that you can use them in batch files or scripts to perform common database operations. You can automate execution of scripts through your operating system's scheduling facility (**cron** on UNIX, **AT** on Windows NT/2000). It is more difficult to automate execution of graphical tools.

isql

The **isql** tool is a shell-type interactive program that enables you to quickly and easily enter SQL statements to execute with respect to a database. This tool uses InterBase's Dynamic SQL mechanism to submit a statement to the server, prepare it, execute it, and retrieve any data from statements with output (for example, from a SELECT or EXECUTE PROCEDURE). **isql** manages transactions, displays metadata information, and can produce and execute scripts containing SQL statements.

See **Chapter 9: "Interactive Query"** for full documentation and reference on **isql** and using **isql** from IBConsole.

gbak

The **gbak** tool provides options for backing up and restoring databases. **gbak** now backs up to multiple files and restores from multiple files, making it unnecessary to use the older **gsplit** command. Only SYSDBA and the owner of a database can back up a database. Any InterBase user defined on the server can restore a database, although the user must be SYSDBA or the database owner in order to restore it *over* an existing database.

Note When you back up and restore databases from IBConsole on Windows platforms, you are accessing this same tool through the IBConsole interface.

See **Chapter 7: "Database Backup and Restore"** for full documentation and reference on using **gbak**.

gfix

gfix configures several properties of a database, including:

- Database active/shutdown status
- Default cache allocation for clients
- Sweep interval and manual sweep
- Synchronous or asynchronous writes
- Detection of some types of database corruption
- Recovery of unresolved distributed transactions

You can also access all the functionality of **gfix** through the IBConsole graphical interface. Only SYSDBA and the owner of a database can run **gfix** against that database.

See **Chapter 6: "Database Configuration and Maintenance"** for descriptions of these properties, and a reference of the **gfix** tool.

gsec

You can configure authorized users to access InterBase servers and databases with **gsec**. You can also perform the same manipulations on the security database with IBConsole.

See **Chapter 5: "Database Security"** for full details and reference.

gstat

gstat displays some database statistics related to transaction inventory, data distribution within a database, and index efficiency. You can also view these statistics from IBConsole. You must be SYSDBA or the owner of a database to view its statistics.

See **Chapter 8: "Database and Server Statistics"** for more information on retrieving and interpreting database statistics.

iblockpr (gds_lock_print)

You can view statistics from the InterBase server lock manager to monitor lock request throughput and identify the cause of deadlocks in the rare case that there is a problem with the InterBase lock manager. The utility is called **gds_lock_print** on the UNIX platforms, and **iblockpr** on the Windows platforms.

See **Chapter 8: "Database and Server Statistics"** for more information on retrieving and interpreting lock statistics.

ibmgr

On UNIX servers, use the **ibmgr** utility to start and stop the InterBase server process. See the section **"Using ibmgr to start and stop the server" on page 44** for details on using this utility.

CHAPTER

2

IBConsole: The InterBase Interface

InterBase provides an intuitive graphical user interface, called IBConsole, with which you can perform every task necessary to configure and maintain an InterBase server, to create and administer databases on the server, and to execute interactive SQL (ISQL). IBConsole enables you to:

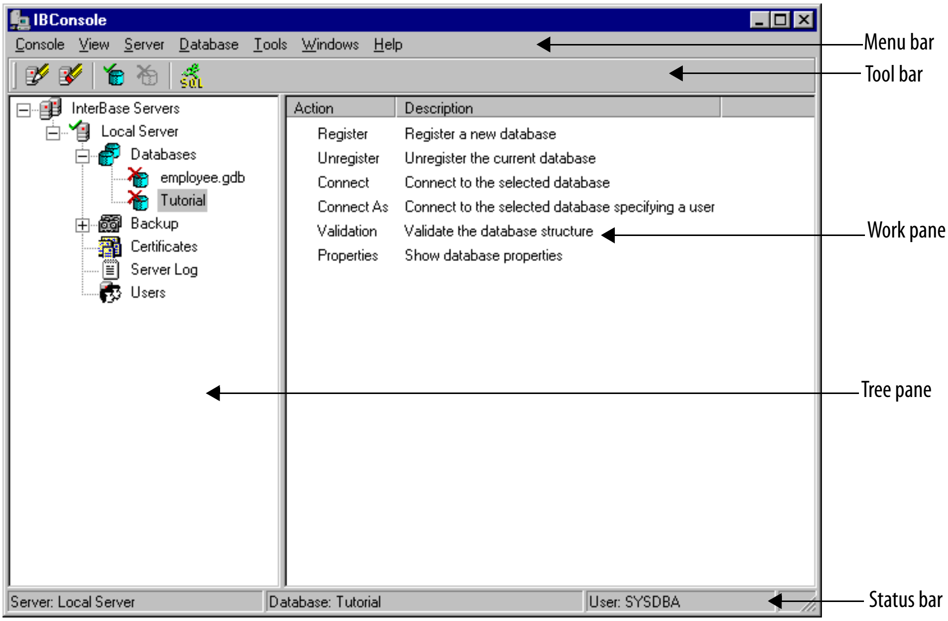
- Manage server security
- Back up and restore a database
- View database and server statistics
- Perform database maintenance, including:
  - Validating the integrity of a database
  - Sweeping a database
  - Recovering transactions that are "in limbo"

IBConsole runs on Windows, but can manage databases on any server on the local network.

Starting IBConsole

To start IBConsole, choose IBConsole from the **Start | Programs | InterBase** menu. The IBConsole window opens:

FIGURE 2.1 IBConsole window



Action	Description
Register	Register a new database
Unregister	Unregister the current database
Connect	Connect to the selected database
Connect As	Connect to the selected database specifying a user
Validation	Validate the database structure
Properties	Show database properties

Server: Local Server | Database: Tutorial | User: SYSDBA

Elements in the IBConsole dialog:

- **Menu bar** Commands for performing database administration tasks.
- **Tool bar** Shortcut buttons for menu commands.
- **Tree pane** Displays a hierarchy of registered servers and databases.
- **Work pane** Displays specific information or allows you to perform activities, depending on what item is currently selected in the Tree pane.
- **Status bar** Shows the current server, user login, and selected database.

IBConsole menus

The IBConsole menus are the basic way to perform tasks with IBConsole. There are seven pull-down menus.

- **Console menu** enables you to exit from IBConsole.
- **View menu** enables you to indicate whether or not IBConsole displays system data and dependencies and to change the display and appearance of items listed in the Work pane.
- **Server menu** enables you to register and un-register a server, log in to and log out of a server, diagnose a server connection, manage user security, add and remove certificates, view the server log file, and view server properties. For more information, see **"Connecting to servers and databases" on page 60**.
- **Database menu** enables you to register and un-register a database, create and drop a database, connect to and disconnect from a database, view database metadata, view a list of users connected to the database, view and set database properties, perform database maintenance, validation, and transaction recovery. For more information, see **"Connecting to servers and databases" on page 60**
- **Tools menu** enables you to add custom tools to the Tools menu and start the interactive SQL window. The interactive SQL window has its own set of menus, which are discussed in **Chapter 9: "Interactive Query"**.
- **Windows menu** enables you to view a list of active IBConsole windows and to manage them. See **"Switching between IBConsole windows" on page 38** for more information.
- **Help menu** enables you to access both IBConsole on-line help and InterBase on-line help.

Context menus

IBConsole also enables you to perform certain tasks with context sensitive popup menus called *context menus*. Tables 2.1 and 2.2 are examples of context menus.

When you right-click a server icon, a context menu is displayed listing actions that can be performed on the selected server.

Popup command	Description
Register	Register the current server.
Un-register	Un-register the current server.
Login	Login to the selected server.
Logout	Logout from the current server.

TABLE 2.1 IBConsole context menu for a server icon

Popup command	Description
Add Certificate	Add certificate ID/keys for the current server.
User Security	Authorize users on the current server.
View Logfile	Display the server log for the current server.
Diagnose Connection	Display database and network protocol communication diagnostics.
Properties	View and update server information for the current server.

TABLE 2.1 IBConsole context menu for a server icon (*continued*)

When you right-click a connected database icon, a context menu is displayed listing actions that can be performed on the database:

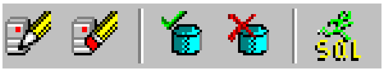
Popup command	Description
Disconnect	Disconnect from the current database.
Maintenance	Perform maintenance tasks including: view database statistics, shutdown, database restart, sweep, and transaction recovery.
Backup/Restore	Back up or restore a database to a device or file.
View Metadata	View the metadata for the selected database.
Connected Users	Displays a list of users connected to the database.
Properties	View database information, adjust the database sweep interval, set the SQL dialect and access mode, and enable forced writes.

TABLE 2.2 IBConsole context menu for a connected database icon

IBConsole toolbar

A toolbar is a row of buttons that are shortcuts for menu commands. The following table describes each toolbar button in detail.

FIGURE 2.2 IBConsole Toolbar



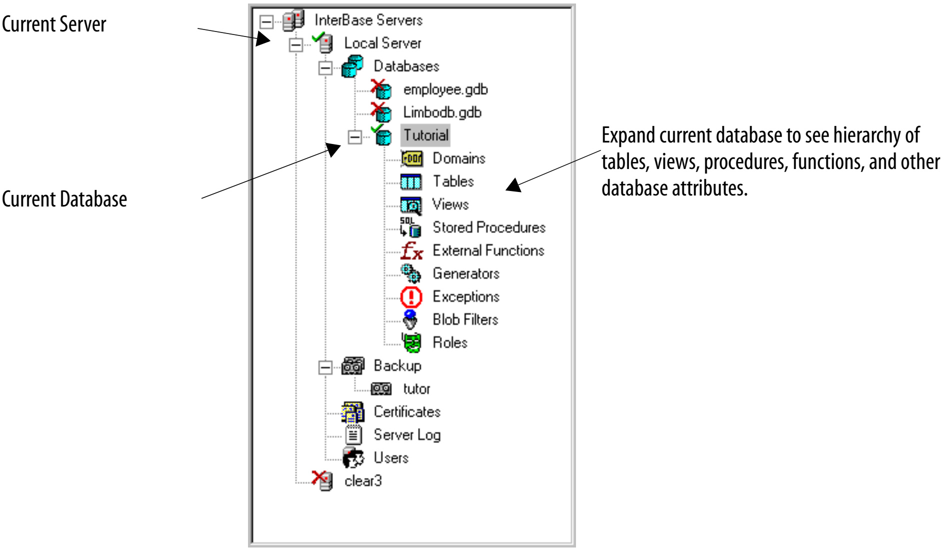
Button	Description
✎	**Register server**: opens the register server dialog, enabling you to register and login to a local or remote server. See “Registering a server” on page 60 for more information.
✎	**Un-register server**: enables you to unregister a local or remote server. This automatically disconnects a database on the server and logout from the server. See “Unregistering a server” on page 63 for more information.
✓	**Database connect**: opens the database connect dialog, enabling you to connect to a database on the current server. See “Connecting to a database” on page 65 for more information.
✗	**Database disconnect**: enables you to disconnect a database on the current server. See “Disconnecting a database” on page 67 for more information.
SQL	**Launch SQL**: opens the interactive SQL window, which is discussed in detail in Chapter 9: “Interactive Query”.

TABLE 2.3 IBConsole standard toolbar

Tree pane

When you open the IBConsole window, you must register and log in to a local or remote server and then register and connect to the server’s databases to display the Tree pane. See **“Connecting to servers and databases” on page 60** to learn how to register and connect servers and databases.

FIGURE 2.3 IBConsole Tree pane



Current Server

Current Database

Expand current database to see hierarchy of tables, views, procedures, functions, and other database attributes.

Navigating the server/database hierarchy is achieved by expanding and retracting nodes (or branches) that have subdetails or attributes. This is accomplished by a number of methods, outlined in Table 2.4.

To expand or retract the server/database tree in the Tree pane:

Tasks	Commands
Display a server's databases	• Left-click the plus (+) to the left of the server icon • Double-click the server icon • Press the plus (+) key • Press the right arrow key
Retract a server's databases	• Left-click the minus (–) to the left of the server icon • Double-click the server icon • Press the minus (–) key • Press the left arrow key

TABLE 2.4 Server/database tree commands

In an expanded tree, click a database name to highlight it. The highlighted database is the one on which IBConsole operates, referred to as the *current database*. The *current server* is the server on which the current database resides.

The hierarchy displayed in the Tree pane of **FIGURE 2.3** is an example of a fully expanded tree.

- Expanding the InterBase Servers branch displays a list of registered servers.
- Expanding a connected server branch displays a list of server attributes, including Databases, Backup, Users, Certificates, and the Server Log.
- Clicking on the Database branch displays a list of registered databases on the current server.
- Clicking on Server Log displays the "View Logfile" action in the Work pane.
- Expanding a connected database branch displays a list of database attributes, including Domains, Tables, Views, Stored Procedures, External Functions, Generators, Exceptions, Blob Filters, and Roles.
- Expanding Backup shows a list of backup aliases.

Work pane

Depending on what item has been selected in the Tree pane, the Work pane gives specific information or enables you to execute certain tasks.

- Clicking on the Backup icon displays a list of backup aliases for the current server.
- Clicking on the Certificates icon displays a list of InterBase certificate keys and IDs for the current server.
- Clicking on the Users icon displays a list of users defined on the server.
- Clicking on a a database attribute icon displays information for that particular attribute. Clicking on the icon for a database object, such as a table name, in the Work Pane launches an object viewer specific to that object. These are discussed in **"Viewing metadata" on page 200**.

Standard text display window

The standard text display window is used to monitor database backup and restoration, to display database statistics and to view server and administration logs.

The standard text display window contains a menu bar, a toolbar with icons for often-used menu commands, and a scrolling text display area. **FIGURE 7.3, "Database backup verbose output," on page 150** is an example of the standard text display window.

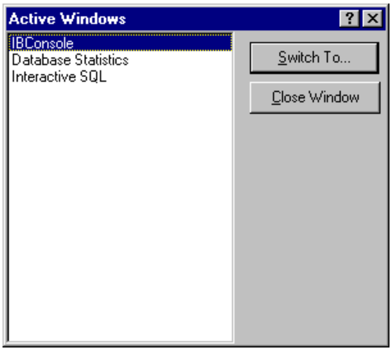
Elements in a standard text display window:

- **Menu bar** The File menu enables you to save the contents of the window and Exit from the window. The Edit menu enables you to copy selected text in the window to the clipboard, select all text in the window, cut and paste text, and find a specified word or phrase within the displayed text.
- **Toolbar** Save and Copy toolbar buttons enable you to save the contents of the text display window as well as copy selected text to the clipboard.
- **Status bar** Shows the cursor location, given by line and column, within the text display window.

Switching between IBConsole windows

Use the Active Windows dialog to switch between IBConsole windows, or to close specific windows. To access the Active Windows dialog, click on the Windows menu. The dialog appears:

FIGURE 2.4 Active Windows dialog



Active Windows

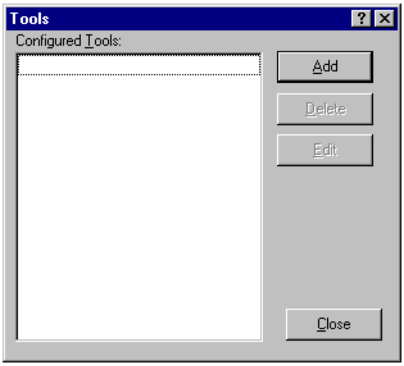
IBConsole
Database Statistics
Interactive SQL

- To switch to a different IBConsole window, select it and click the Switch To button.
- To close a window, select it and click the Close window button.

Managing custom tools in IBConsole

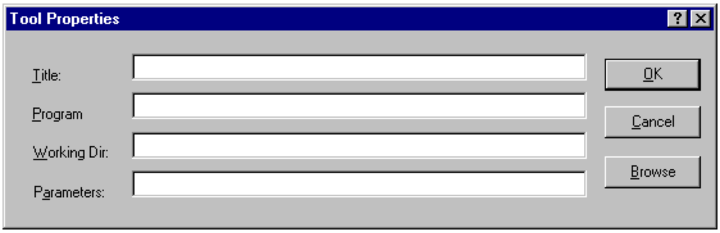
Use the Tools dialog to add, edit, and delete custom tools for the IBConsole interface. To access the Tools dialog, select **Tools | Configure Tools** from the IBConsole menu. The Tools dialog is displayed:

FIGURE 2.5 Tools dialog



- To delete a tool, select it and click Delete.
- To modify a tool, select it and click Edit. Change the relevant fields in the **Tool Properties dialog**.
- To add a tool, click Add. The Tool Properties dialog appears.

FIGURE 2.6 Tool Properties dialog



To add a custom tool:

- Enter the name of your utility in the Title field. This is the name that will be displayed on the Tools menu. Use an ampersand (&) to specify an accelerator key for the menu item. Conflicting accelerator keys are automatically resolved. If you do not specify an accelerator key, one will be chosen automatically.
- Enter the path and the executable to be launched in the Program field.
- Enter the working directory for your utility in the Working Dir field. If no working directory is specified, then it defaults to the current directory.
- Enter any other parameters needed to run your utility in the Parameters field.

CHAPTER

3

Server Configuration

This chapter describes the operation and configuration of the InterBase server process, including the following topics:

- **Configuring server properties**
- **Using InterBase Manager to start and stop InterBase**
- **Starting and stopping the InterBase Server on UNIX**
- **Example initialization script installation on Linux**
- **Using environment variables**
- **Managing temporary files**
- **Configuring parameters in isc_config**
- **Viewing the server log file**

Configuring server properties

You can use InterBase Guardian to change database cache size of client map size. The InterBase Guardian Server Properties dialog enables you to display and configure these server settings. To access InterBase Guardian, right-click the InterBase Guardian icon in the System Tray. You can access the Server Properties dialog by any of the following methods:

- Select a server (or any branch under the server hierarchy) in the Tree pane and choose **Server | Server Properties**.
- Select a server in the Tree pane and click Server Properties in the Work pane.
- Right-click a server in the Tree pane and choose Server Properties from the context menu.

The Server Properties dialog contains two tabs, General and IB Settings.

The General tab

The General tab of the Server Properties dialog is where you can view such server settings as the version, capabilities, number of databases, and number of attachments. You cannot edit the information displayed on this tab.

The server properties displayed are:

- **Version**: displays the version number for the InterBase Server.
- **Capabilities**: displays support capabilities for the InterBase Server.
- **Number of databases**: displays the total number of databases in the InterBase Server.
- **Number of attachments**: displays the total number attachments to the InterBase Server.

You cannot update the information displayed on the General tab; however, you can click Refresh at any time to retrieve the current server property information. If you need to view or configure server settings, click the IB Settings tab .

The IB Settings tab

The IB Settings tab of the Server Properties dialog allows you to configure the database cache and client map size. With the IB Settings window open, click **Modify** to enable these fields.

IMPORTANT You must supply the SYSDBA password to gain access to these settings.

Once these fields are enabled, the **Modify** button changes to **Reset**. Clicking **Reset** restores the default values. Below are explanations of the configurable options.

▸ *Database cache*

Database cache is the number of memory pages reserved for each attached database. If the figure is set high enough to accommodate the page requirements for all attached databases, overall performance is maximized because all database activity can be handled in physical RAM rather than having it swapped to disk. If too many pages are reserved, however, and you have many databases running simultaneously, your request might exceed the amount of physical RAM available to the system. If that happens, some of your operations would be swapped to disk as the operating system tries to manage the excessive demands of your databases and the needs of other running applications (including itself).

To calculate the amount of memory required by a database, multiply the number of pages by the page size (these figures are defined when a database is created).

Default page allocation is 75 for each running database. Minimum is 50 pages. There is no maximum, but the total allocation must not exceed system resources. New settings take effect as soon as you click **OK** or **Apply**.

▸ *Client map size*

Client map size is the size in bytes of the communication buffer for each InterBase client (local clients only). You might want to increase this setting when dealing with retrieval of large data sets such as graphic blobs.

Default size is 4KB, but the range is 1KB to 8KB. New settings take effect the next time the server is started.

Using InterBase Manager to start and stop InterBase

The InterBase Server and InterBase Guardian must be started before you enable database connections. Use the InterBase Manager on Windows platforms to start and stop the InterBase Server and Guardian. To start the InterBase Manager, choose **Start | Settings | Control Panel | InterBase Manager**; InterBase Manager is a Windows Control Panel applet. You can use InterBase Manager to do the following:

- Choose the server startup mode: whether to start the InterBase server manually, or have it start automatically at system boot time
- Change the path to the server: if you click the **Change** option, you can browse and select a different directory
- Change how InterBase Server operates. By default, InterBase runs automatically as a service on Windows NT or Windows 2000, though it is possible (but not recommended) to run it as an application. On Windows 98, InterBase runs only as an application.

Note To start InterBase Server as an application from a command prompt or in a batch file, invoke InterBase Guardian with option -a:

ibguard -a

InterBase Guardian starts the server, and also places an icon in the System Tray.

- **Start** InterBase Server and InterBase Guardian, via a Start/Stop button. Click **Start** in the InterBase Manager Status area to start InterBase Server (and InterBase Guardian). The server status changes, and an InterBase Guardian icon appears in the system tray. Once you have started the InterBase Server, you can exit InterBase Manager, and both InterBase Server and InterBase Guardian will continue to run. The InterBase Guardian icon remains in the System Tray until you stop the server.
- **Stop** InterBase Server and InterBase Guardian, via a Start/Stop button. Click **Stop** in the InterBase Manager Status area to stop InterBase Server (and InterBase Guardian). Or, right-click the InterBase Guardian icon in the System Tray and choose **Shutdown**.

Starting and stopping the InterBase Server on UNIX

The following sections describe how to start and stop the InterBase server on UNIX.

Using ibmgr to start and stop the server

The InterBase Server process **ibserver** runs as a daemon on UNIX systems. Use **ibmgr** to start and stop the server process. To use **ibmgr**, you must be logged on to the server machine.

Syntax

```
ibmgr -command [-option [parameter] ...]
```

or

```
ibmgr [ F5 → PgDn F5 Del  ↑ F5 Alt ]
IBMGR> command [-option [parameter]]
```

Description On UNIX, the InterBase server process runs as a daemon. A daemon runs even when no user is logged in to the console or a terminal on the UNIX system.

ibmgr is a utility for managing the InterBase server process on UNIX systems. You must be logged on to the machine on which the server is running to use **ibmgr**.

Note The **ibmgr32.exe** file that is present in older Windows installations is an older client-side utility whose functions are entirely different than **ibmgr** on UNIX. The name is coincidental.

Options

start [**–once** | **–forever**]	Starts server; the **–forever** switch causes the server to restart if it crashes; default is **–forever**
shut	Rolls back current transactions, terminates client connections, and shuts down server immediately
show	Shows host and user
user *user_name*	Supplies SYSDBA
password *password*	Supplies SYSDBA password
help	Prints help text
quit	Quits prompt mode

TABLE 3.1 **ibmgr** commands

Starting the server

To start the InterBase server, log in as the “root” or “interbase” user. (“interbas” is a synonym for “interbase,” to accommodate operating systems that do not support nine-character account names.) For example, start InterBase with the following command:

```
ibmgr -start
```

Note Once you have started **ibserver** using one login, such as “root,” be aware that all objects created belong to that login. They are not accessible to you if you later start **ibserver** as one of the other two (“interbas” or “interbase”). It is highly recommended to run the InterBase Server as “interbase.”

InterBase Classic uses the **inetd** process to handle incoming requests. There is no need to explicitly start the server; **inetd** forks off a process to handle incoming requests. Usually, **inetd** is set up to start automatically.

Stopping the server

Note For safety, make sure all databases have been disconnected before you stop the server.

The command switches **–user** and **–password** can be used as option switches for commands like **–start** or **–shut**. For example, you can shut down a server in any of the following ways:

```
ibmgr -shut -password password
```

or

```
ibmgr Enter
IBMGR> shut -password password
```

or

```
imbgr Enter
IBMGR> password password
IBMGR> shut
```

Note The **–shut** option rolls back all current transactions and shuts down the server immediately. If you need to allow clients a grace period to complete work and detach gracefully, use shutdown methods for individual databases. See **"Shutting down and restarting databases" on page 131**.

Starting the server automatically

To configure a UNIX server to start the InterBase Server automatically at server host boot-time, you must install a script that the **rc** initialization scripts can run. Refer to **/etc/init.d/README** for more details on how UNIX runs scripts at boot-time.

Example initialization script

```
#!/bin/sh
# ibserver.sh script - Start/stop the InterBase daemon

# Set these environment variables if and only if they are not set.
: ${INTERBASE:=/usr/interbase}
# WARNING: in a real-world installation, you should not put the
# SYSDBA password in a publicly-readable file. To protect it:
# chmod 700 ibserver.sh; chown root ibserver.sh
export INTERBASE

ibserver_start() {
   # This example assumes the InterBase server is
   # being started as UNIX user ’interbase’.
   echo '$INTERBASE/bin/ibmgr -start -forever' | su interbase
}
```

```
ibserver_stop() {
    # No need to su.
    $INTERBASE/bin/ibmgr -shut -user SYSDBA -password password
}

case $1 in
’start’) ibserver_start ;;
’start_msg’) echo 'InterBase Server starting...\c' ;;

’stop’) ibserver_stop ;;
’stop_msg’) echo 'InterBase Server stopping...\c' ;;

*) echo 'Usage: $0 { start | stop }' ; exit 1 ;;
esac

exit 0
```

Example initialization script installation on Solaris

1. Log in as root.

```
$ su
```

2. Enter the example script above into the initialization script directory.

```
# vi /etc/init.d/ibserver.sh
```

3. Enter text
4. Link the initialization script into the **rc** directories for the appropriate run levels for starting and stopping the InterBase server.

```
# ln /etc/init.d/ibserver.sh /etc/rc0.d/K30ibserver
# ln /etc/init.d/ibserver.sh /etc/rc2.d/S85ibserver
```

Example initialization script installation on HP-UX

1. Log in as root.

```
$ su
```

2. Enter the example script above into the initialization script directory.

```
# vi /sbin/init.d/ibserver.sh
<Enter text>
```

3. Link the initialization script into the **rc** directories for the appropriate run levels for starting and stopping the InterBase server.

```
# ln -s /sbin/init.d/ibserver.sh /sbin/rc1.d/K500ibserver
# ln -s /sbin/init.d/ibserver.sh /sbin/rc2.d/S500ibserver
```

Example initialization script installation on Linux

1. Log in as root.

```
$ su
```

2. Enter the Linux example script (given below) into the initialization script directory.

```
# cp ibserver.sh /etc/rc.d/init.d/ibserver.sh
# chmod 700 /etc/rc.d/init.d/ibserver.sh
```

3. Link the initialization script into the **rc** directories for the appropriate run levels for starting the InterBase server.

```
# ln -s /etc/rc.d/init.d/ibserver.sh /etc/rc.d/rc0.d/S85ibserver
```

4. Link the initialization script into the **rc** directories for the appropriate run levels for stopping the InterBase server.

```
# ln -s /etc/rc.d/init.d/ibserver.sh /etc/rc.d/rc0.d/K30ibserver
```

5. Make sure you have host equivalence:

```
# touch /etc/gds_hosts.equiv
# echo “+” >> /etc/gds_hosts.equiv
```

6. Make sure you don’t have an **inetd** entry for InterBase Classic:

```
# echo -e “/gds_db/s/^/#/\nwq” | ed /etc/inetd.conf
# killall -HUP inetd
```

Example Linux initialization script

```
#!/bin/sh
# ibserver.sh script - Start/stop the InterBase daemon
# Set these environment variables if and only if they are not set.
: ${INTERBASE:=/usr/interbase}
# WARNING: in a real-world installation, you should not put the
# SYSDBA password in a publicly-readable file. To protect it:
# chmod 700 ibserver.sh; chown root ibserver.sh
export INTERBASE

ibserver_start() {
    # This example assumes the InterBase server is
    # being started as user “interbase”.
    su - interbase -c “$INTERBASE/bin/ibmgr -start -forever”
    RETVAL=$?
    [ $RETVAL -eq 0 ] && touch /var/lock/subsys/ibserver
}

ibserver_stop() {
    # No need to su.
    $INTERBASE/bin/ibmgr -shut -user SYSDBA -password password
    RETVAL=$?
    [ $RETVAL -eq 0 ] && rm -f /var/lock/subsys/ibserver
}

if [ ! -d “$INTERBASE” ] ; then
    echo “$0: cannot find InterBase installed at $INTERBASE” >&2
    exit 1
fi
if [ ! -x “$INTERBASE/bin/ibmgr” ] ; then
    echo “$0: cannot find the InterBase SuperServer manager as
                    $INTERBASE/bin/ibmgr” >&2
if [ ! -x “$INTERBASE/bin/gds_inet_server” ] ; then
    echo “$0: this is InterBase Classic; use inetd instead of
                    ibserver daemon” >&2
    fi
    exit1
fi

case $1 in
’start’)
    ibserver_start ;
    echo “InterBase started” ;;
’start_msg’)
    echo 'InterBase Server starting...\c' ;;

’stop’)
    ibserver_stop ;
    echo “InterBase stopped” ;;
’stop_msg’)
    echo 'InterBase Server stopping...\c' ;;

’restart’)
    ibserver_stop ; ibserver_start
    echo “InterBase restarted” ;;
’restart_msg’)
```

```
    echo 'InterBase Server restarting...\c' ;;

*) echo "Usage: $0 { start | stop | restart }" ; exit 1 ;;
esac

exit 0
```

The attachment governor

The InterBase server has an attachment governor that regulates the number of attachments to the server. Multiply the value of the USERS field in the license file by four to determine the total number of permitted concurrent attachments.

All successful attempts to create or connect to a database increment the number of current attachments. Both local and remote connections count toward the connection limit. Connections are permitted by the governor until the maximum number of concurrent attachments is reached. All successful attempts to drop or disconnect from a database decrement the number of current attachments.

Once the maximum number of attachments is reached, the server returns the error constant *isc_max_att_exceeded* (defined in **ibase.h**), which corresponds to the message:

```
Maximum user count exceeded. Contact your database administrator.
```

Using environment variables

This section describes the usage of environment variables that InterBase recognizes. When defining environment variables, keep these rules in mind:

- Environment variables must be in the scope of the **ibserver** process (SuperServer architecture) or **gds_inet_server** process (Classic architecture).
- On Windows NT/2000, define environment variables as *system variables* in the Windows **Control Panel | System | Environment** dialog.
- On UNIX, the easiest way to define environment variables is to add their definitions to the system-wide default shell profile.

ISC_USER and ISC_PASSWORD

If you do not provide a user name and password when you connect to a database or when you run utilities such as **gbak**, **gstat**, and **gfix**, InterBase looks to see if the ISC_USER and ISC_PASSWORD environment variables are set; if they are, InterBase uses that user and password as the InterBase user.

Although setting these environment variables is convenient, do not do so if security is at all an issue.

The INTERBASE environment variables

INTERBASE

The INTERBASE variable is used both during installation and during runtime. During installation, it defines the path where the InterBase product is installed. If this path is different from **/usr/interbase**, all users must have the correct path set at runtime. During runtime, use the INTERBASE variable to set the InterBase install directory.

INTERBASE_TMP

The INTERBASE_TMP variable can be used to set the location of InterBase's sort files on the server. There are other options for defining the location of these files. See **"Configuring sort files" on page 52**.

INTERBASE_LOCK AND INTERBASE_MSG

INTERBASE_LOCK sets the location of the InterBase lock file and INTERBASE_MSG sets the location of the InterBase message file. These two variables are independent of each other and can be set to different locations.

IMPORTANT The environment variables must be in the scope of the **ibserver** process. On Windows platforms, define the variables as *system variables* in the **Control Panel | System | Environment** dialog. On UNIX, the easiest way to do this is to add the variable definition to the system-wide default shell profile.

The TMP environment variable

The TMP environment variable defines where InterBase stores temporary files, if the INTERBASE_TMP variable is not defined.

Managing temporary files

InterBase creates two types of temporary files: sort files and history list files.

The InterBase server creates sort files when the size of the internal sort buffer isn't big enough to perform the sort. Each request (for example, CONNECT or CREATE DATABASE) gets and shares the same list of temporary file directories. Each request creates its own temporary files (each has its own I/O file handle). Sort files are released when sort is finished or the request is released. If space runs out in a particular directory, InterBase creates a new temporary file in the next directory from the directory list. If there are no more entries in the directory list, it prints an error message and stops processing the current request.

The InterBase **isql** client creates the history list files to keep track of the input commands. Each instance creates its own temporary files, which can increase in size until they run out of disk space. Temporary file management is not synchronized between clients. When a client quits, it releases its temporary files.

Configuring history files

To set the location for history files, define the TMP environment variable on your client machine. This is the *only* way to define the location of history files. If you do not set the location for the history files by defining the TMP environment variable, an InterBase client uses whatever temporary directory it finds defined for the local system. If no temporary directory is defined, it uses **/tmp** on a UNIX system or **C:\temp** on a Windows system.

Configuring sort files

You should make sure to have plenty of free space available for temporary sorting operations. The maximum amount of temporary space InterBase needs might be larger than the database itself in some cases.

Temporary sort files are always located on the server where the database is hosted; you should specify temporary directories on disk drives that are physically local to the server (not on mapped drives or network mounted filesystems).

There are two ways to specify directories for sort files:

- You can add an entry to the **$INTERBASE/isc_config** (UNIX) or **ibconfig** (Windows) file to enable directory and space definition for sort files. The syntax is:

```
TMP_DIRECTORY size "pathname"
```

IMPORTANT The pathname **must** be in double quotes, or the config file will fail.

This defines the maximum size in bytes of each sort directory. You can list several directories, each on its own line with its own size specification and can specify a directory more than once with different size configurations. InterBase exhausts the space in each specification before proceeding to the next one.

For example, if you specify **dir1** with a size of 5,000,000 bytes, then specify **dir2** with 10,000,000 bytes, followed by **dir1** with 2,000,000 bytes, InterBase uses **dir1** until it reaches the 5,000,000 limit, then uses **dir2** until it has filled the 10,000,000 bytes allocated there, and then returns to **dir1** where it has another 2,000,000 bytes available. Below are the **ibconfig** entries that describe this configuration:

```
TMP_DIRECTORY 5000000 "C:\dir1"
TMP_DIRECTORY 10000000 "D:\dir2"
TMP_DIRECTORY 2000000 "C:\dir1"
```

- You can use the INTERBASE_TMP and TMP environment variables to define the location.

If you specify temporary directories in **isc_config** (UNIX) or **ibconfig** (Windows), the server uses those values for the sort files and ignores the server environment variable values. If you don't specify configuration of temporary directories in **isc_config** or **ibconfig**, then the server picks a location for a sort file based on the following algorithm:

1. Use the directory defined in INTERBASE_TMP environment variable
2. If INTERBASE_TMP isn't defined, use directory defined in TMP environment variable
3. If TMP isn't defined, default to the **/tmp** directory (UNIX) or **C:\temp** (Windows)

Configuring parameters in isc_config

You specify configuration parameters for InterBase Server by entering their names and values in a text file: **isc_config** on UNIX, or **ibconfig** on Windows. Entries are in the form:

```
parameter <whitespace> value
```

- *"parameter"* is a string that contains no whitespace and names a property of the server being configured.
- "*value*" is a number or string that is the configuration of the specified property.

Note Each line in **isc_config** is limited to 80 characters, including the word "parameter" and any whitespace.

The following is a summary of the legal entries in **isc_config** (UNIX) or **ibconfig** (Windows) file:

Parameter	Description
connection_timeout	Seconds to wait before concluding an attempt to connect has failed; default is 180.
database_cache_pages	Server-wide default for the number of database pages to allocate in memory per database. This can be overridden by clients. See **"Configuring the database cache" on page 124** for more information. Defaults: 2048 - SuperServer, 75 - Classic.
deadlock_timeout	Seconds before an ungranted lock causes a scan to check for deadlocks; default is 10.
dummy_packet_interval	Seconds to wait on a silent client connection before the server sends dummy packets to request acknowledgment; default is 60.
lock_acquire_spins	Number of spins during a busy wait on the lock table mutex. Only relevant on SMP machines; default is 0.
lock_hash_slots	Tune lock hash list. More hash slots means shorter hash chains. Not necessary except under very high load. Prime number values are recommended; default is 101.
server_client_mapping	Size in bytes of one client's portion of the memory mapped file used for interprocess communication; default is 4096.
server_priority_class	Priority of the InterBase service on Windows NT or Windows 2000. The value 1 is low priority, 2 is high priority. Relevant on Windows NT/2000 only; default is 1.
server_working_size_max	Threshold above which the OS is requested to swap out all memory. Relevant on Windows NT and Windows 2000 only; default is 0 (system-determined).
server_working_size_min	Threshold below which the OS is requested to swap out no memory. Relevant on Windows NT and Windows 2000 only; default is 0 (system-determined).
tmp_directory	Directory to use for storing temporary files. Specify number of bytes available in the directory, and the path of the directory. You can list multiple entries, one per line. Each directory is used according to the order specified; default is the value of the INTERBASE_TMP environment variable, otherwise **/tmp** on UNIX or **C:\temp** on Windows NT/2000.
v4_event_memsize	Bytes of shared memory allocated for event manager; default is 32768.

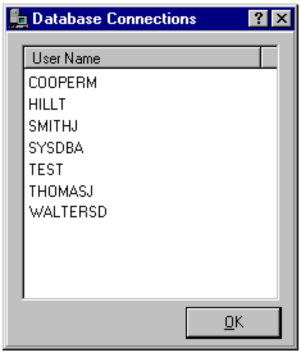
Parameter	Description
v4_lock_grant_order	1 means locks are granted first come, first served. 0 means emulate InterBase V3.3 behavior, where locks are granted as soon as they are available, which can result in lock request starvation; default is 1.
v4_lock_mem_size	Bytes of shared memory allocated for lock manager; default is 98304.
v4_lock_sem_count	Number of semaphores for interprocess communication. Classic architecture only; default is 32.
v4_lock_signal	UNIX signal to use for interprocess communication. Classic architecture only; default is 16.
v4_solaris_stall_value	Number of seconds a server process waits before retrying for the lock table mutex. Relevant on Solaris only; default is 60.

Monitoring client connections with IBConsole

You can view current user activity on a particular database in IBConsole using the Database Connections dialog. You can access this dialog by one of the following methods:

- Select a database (or any branch under the database hierarchy) in the Tree pane and choose **Database | Maintenance | Connections**.
- Select a database in the Tree pane and click View Current Connections in the Actions tab of the Work pane.
- Right-click a database in the Tree pane and choose Connections from the context menu.
- Select Database Aliases in the Tree pane to display a list of registered databases in the Summary tab of the Work pane. Right-click a database in the Work pane and choose Connections from the context menu.

FIGURE 3.1 Database connections dialog


Database Connections

User Name

COOPERM
HILLT
SMITHJ
SYSDBA
TEST
THOMASJ
WALTERSD

This dialog displays the list of active users connected to the database.

Viewing the server log file

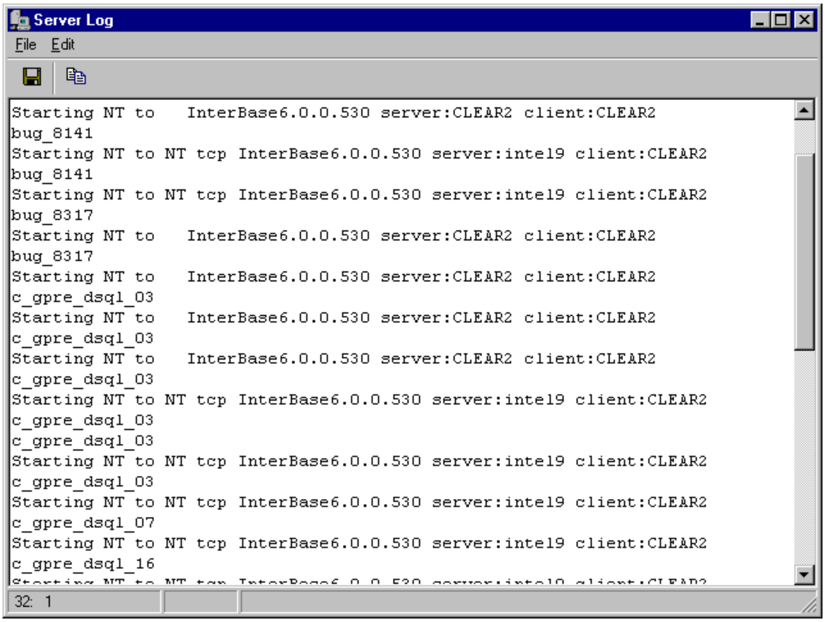
InterBase Server logs diagnostic messages in the file **interbase.log** in the InterBase install directory. Any messages generated by **ibserver** are sent to this file. This can be a very important source of diagnostic information if your server is having configuration problems.

Refer to the *Language Reference* for a list of error messages that can appear in this file.

IBConsole displays this log file in a standard text display window. To display the Server Log dialog:

- Select a server and expand it if it is not already expanded, click on Server Log and then click on View Logfile in the Work pane.
- Right-click a server in the Tree pane and choose View Logfile from the context menu.
- Select a server and then choose View Logfile from the Server menu.

FIGURE 3.2 Server Log dialog



```
Starting NT to   InterBase6.0.0.530 server:CLEAR2 client:CLEAR2
bug_8141
Starting NT to NT tcp InterBase6.0.0.530 server:intel9 client:CLEAR2
bug_8141
Starting NT to NT tcp InterBase6.0.0.530 server:intel9 client:CLEAR2
bug_8317
Starting NT to   InterBase6.0.0.530 server:CLEAR2 client:CLEAR2
bug_8317
Starting NT to   InterBase6.0.0.530 server:CLEAR2 client:CLEAR2
c_gpre_dsql_03
Starting NT to   InterBase6.0.0.530 server:CLEAR2 client:CLEAR2
c_gpre_dsql_03
Starting NT to   InterBase6.0.0.530 server:CLEAR2 client:CLEAR2
c_gpre_dsql_03
Starting NT to NT tcp InterBase6.0.0.530 server:intel9 client:CLEAR2
c_gpre_dsql_03
c_gpre_dsql_03
Starting NT to NT tcp InterBase6.0.0.530 server:intel9 client:CLEAR2
c_gpre_dsql_03
Starting NT to NT tcp InterBase6.0.0.530 server:intel9 client:CLEAR2
c_gpre_dsql_07
Starting NT to NT tcp InterBase6.0.0.530 server:intel9 client:CLEAR2
c_gpre_dsql_16
Starting NT to NT tcp InterBase6.0.0.530 server:intel9 client:CLEAR2
```

The standard text display window enables you to search for specific text, save the text to a file, and print the text. For an explanation of how to use the standard text display window, see **"Standard text display window" on page 37.**

CHAPTER

4

Network Configuration

This chapter details issues with configuring InterBase in a networked client/server environment. Topics include network protocols supported by InterBase, remote connection specifiers, and network troubleshooting tips.

Network protocols

InterBase supports TCP/IP for all combinations of client and server platforms. Additionally, InterBase supports IPX/SPX for NetWare servers, NetBEUI for NT/2000 servers and Windows clients, and a *local* connection mode (involving interprocess communication but no network interface) for Windows clients.

InterBase is designed to allow clients running one operating system to access an InterBase server that is running on a different platform and operating system than the client. For example, a common arrangement is to have several Windows 98 PCs acting as client workstations concurrently accessing a departmental server running Windows 2000.

	Server platform			
Client platform	**Windows 98/ME server**	**Windows NT/2000 server**	**UNIX server**	**NetWare server**
Windows 98/ME	TCP/IP, Local	TCP/IP, NetBEUI	TCP/IP	TCP/IP, IPX/SPX
Windows NT, Windows 2000	TCP/IP	TCP/IP, NetBEUI, Local	TCP/IP	TCP/IP, IPX/SPX
UNIX/Linux	TCP/IP	TCP/IP	TCP/IP	TCP/IP

TABLE 4.1 Matrix of connection supported protocols

Note The InterBase client does not support IPX/SPX, though it can use TCP/IP to connect to a NetWare server. To use IPX/SPX, you must use the InterBase 5.1 or higher client.

Connecting to servers and databases

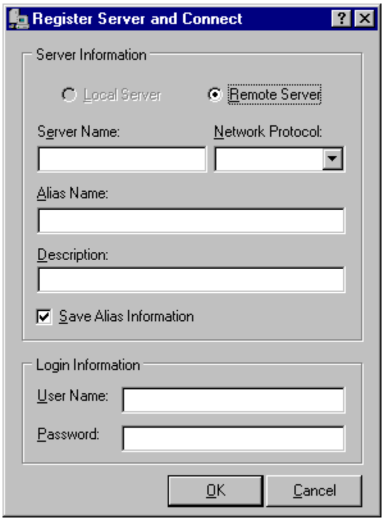
Before performing any database administration tasks, you must first register and log in to a server. Once you log in, you can register and connect to databases residing on the server. You can switch context from one connected database to another by selecting the desired database from the IBConsole Tree pane. The selected database in the Tree pane is referred to as the *current database*. The selected server or the server where the current database resides is referred to as the *current server*.

Registering a server

You can access the Register Server and Connect dialog in IBConsole by one of the following methods:

- Choose **Server | Register** or click the Register Server toolbar button.
- Double-click InterBase Servers in the Tree pane.
- Right-click InterBase Servers or any server in the Tree pane and choose Register from the context menu.

FIGURE 4.1 Register Server and Connect dialog



○ Local Server ● Remote Server

☑ Save Alias Information

To register a local or remote server:

1. Select either the Local Server option or the Remote Server option.
2. If you choose Local Server, the Server Name, Network Protocol and Alias Name information is not required. These text fields are disabled. You can proceed to step 5.
3. If you choose Remote Server, type the name of the server in the Server Name text field, select a network protocol from the drop down list, and enter a server alias name in the Alias Name text field. Check the Save Alias Information check box if you wish to save the server alias name in the windows registry.

   Note The InterBase server name is the name of the database server machine. There is not a specific name for the InterBase server process itself. For example, if the server is running on the NT server "venus", you enter this name in the Server Name text field.

   The network protocol you select can be one of TCP/IP, NetBEUI, or SPX. Protocols are valid only when they are supported by both the client and the server.
4. Optionally, enter a description name for the server.
5. At this point you can choose to just register the server (without logging in) or you can choose to register and connect to the server simultaneously.

- If you want to just register the server you can ignore the Login Information and click OK.
- If you want to register and connect to the server simultaneously, enter a username and password in the corresponding text fields and click OK.

Note The usernames and passwords must be the InterBase usernames and passwords stored in the security database **isc4.gdb** on the server.

Once a server is registered, IBConsole displays it in the Tree pane.

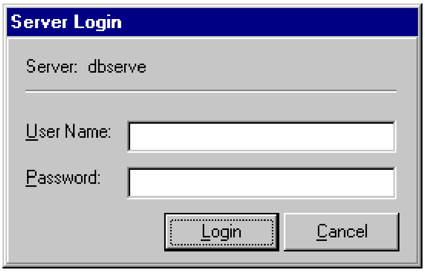
Logging in to a server

You can access the Server Login dialog in IBConsole by one of the following methods:

- In the Tree pane, select a registered server that is not already logged in. Choose **Server | Login** or select Login in the Work pane.
- In the Tree pane, double-click a registered server that is not already logged in.
- In the Tree pane, right-click a registered server that is not already logged in and choose Login from the context menu.

The Server Login dialog appears:

FIGURE 4.2 Server Login dialog



To log in to a server:

1. Verify that the server displayed in the Server field is correct.
2. Enter a username and password in the corresponding text fields. For convenience, IBConsole defaults the UserName text field to the last username that was used to log in (successfully or unsuccessfully).

   Note The usernames and passwords must be the InterBase usernames and passwords that are stored in the security database **isc4.gdb** on the server.

The username is significant to 31 characters and is not case-sensitive. The password is significant to eight characters and is case-sensitive.

All users must enter their username and password to log in to a server. The username and password are verified against records in the security database. If a matching record is found, the login succeeds.

3. Click Login to log in to the server.

IMPORTANT Initially, every server has only one authorized user with username SYSDBA. The SYSDBA must log on and add other authorized users. For more information about how to add new users, see **"User administration with IBConsole" on page 89**.

Logging out from a server

You can log out from a server in IBConsole by one of the following methods:

- Select a connected server in the Tree pane (you can also select any branch under the desired server hierarchy) and choose **Server|Logout**.
- Select a connected server in the Tree pane and click Logout in the Work pane.
- Right-click a connected server in the Tree pane and choose Logout from the context menu.

A confirmation dialog asks you to confirm that you wish to close the connection to the selected server. Click OK if you want to log out from the server, otherwise click Cancel.

Note Logging out from a server automatically disconnects all databases but does not un-register any databases on the server.

Unregistering a server

You can unregister a disconnected server in IBConsole by one of the following methods:

- Select a server in the Tree pane and choose **Server|Un-register** or click the Unregister Server toolbar button
- Select a server in the Tree pane and click Un-register Server in the Work pane.
- Right-click a server in the Tree pane and choose Un-register from the context menu.

A confirmation dialog asks you to confirm that you wish to un-register the selected server. Click OK if you want to un-register the server, otherwise click Cancel.

Note Un-registering a server removes that server from the Tree pane and automatically logs you out of the current server as well as disconnects and un-registers any databases on the server.

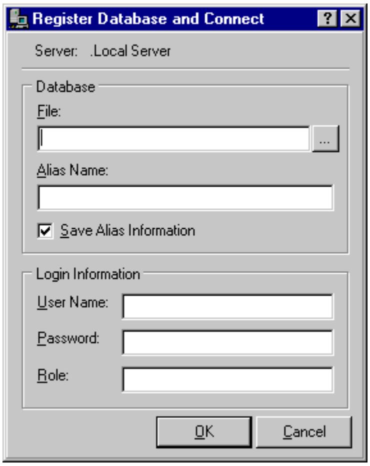
Registering a database

You can access the Register Database and Connect dialog in IBConsole by one of the following methods:

- Choose **Database | Register**.
- Expand a connected server branch. Right-click Databases in the Tree pane and choose Register from the context menu.
- Select a disconnected database in the Tree pane and click Register in the work pane, or right-click the database and choose Register from the context menu.

The Register Database and Connect dialog appears:

FIGURE 4.3 Register Database and Connect dialog



Register Database and Connect

Server: .Local Server

☑ Save Alias Information

To register a database:

1. Make sure the server displayed in the Server field is correct.
2. Enter the database filename, including the path where the file is located, in the File text field. For databases that reside on the local server, you also have the option of clicking the Browse button ... to locate the file you want. The Browse button is disabled for all remote servers.

3. Type an alias name for the database in the Alias Name text field. This is the name that will appear in the IBConsole window. If you omit this step, the alias defaults to the filename that you select in step 2.
4. Check the Save Alias Information check box if you wish to permanently register the database. This saves the database alias name in the windows registry.
5. At this point you can choose to just register the database without connecting, or you can choose to register and connect to the database simultaneously.

   If you only want to register the database, ignore the Login Information and click OK.
6. If you want to register and connect a database simultaneously, type the username, password and optional role for the database in the corresponding text fields and click OK.

   Note If you want to connect using a role, specify the role in the Role text field. This is optional. Connecting using a role gives you all privileges that have been assigned to that role, assuming that you have previously been granted that role with the GRANT statement. For more information on roles, refer to **"ANSI SQL 3 roles" on page 86**.

Once you register a database it appears in the Tree pane.

Connecting to a database

IBConsole provides two methods for connecting to a database. The first method is a quick connect using the username and password that were supplied with the login to the server to instantaneously connect the database. The second method allows you to connect to the database using a different username and password by accessing the Database Connect dialog.

▸ *Connect*

If you want to perform an automatic connect, using the username and password supplied for the server login to instantaneously connect the database, you can do so by one of the following methods:

- Select a disconnected database in the Tree pane. Choose **Database|Connect**, choose Connect in the Work pane, or click on the Database Connect toolbar button.
- Right-click a disconnected database in the Tree pane and choose Connect from the context menu.
- Double-click a disconnected database in the Tree pane.

Once you connect to a database, the database tree expands to display the database hierarchy.

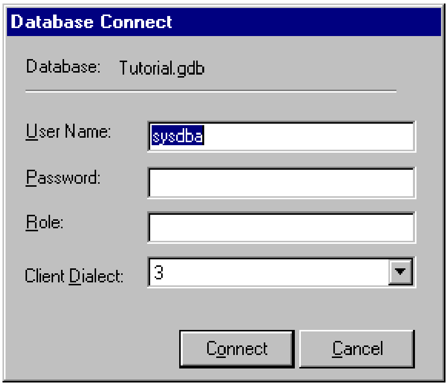
Connect as

If you want to access the Connect Database dialog in IBConsole to connect to the database using a different username and password from that which was supplied in the server login you can do so by one of the following methods:

- Select a disconnected database in the Tree pane. Choose **Database | Connect As** or choose Connect As in the Work pane.
- Right-click a disconnected database in the Tree pane and choose Connect As from the context menu.

The Database Connect dialog appears:

FIGURE 4.4 Database connect dialog



Database: Tutorial.gdb

User Name: sysdba

Password:

Role:

Client Dialect: 3

To connect to a database:

1. Verify that the database displayed in the Database field is correct.
2. Type the username and password for the database in the corresponding User Name and Password text fields.
3. If you want to connect as a role, specify the role in the Role text field. This is optional. Connecting as a role gives you all privileges that have been assigned to that role, assuming that you have previously been granted that role with the GRANT statement. For more information on roles, refer to **"ANSI SQL 3 roles" on page 86**

4. Select the SQL Client dialect. The dialect for the database connection will default to the lower value of the client or server. For more information on SQL dialects, refer to “Understanding SQL Dialects” in *Getting Started*.
5. Click Connect.

Once you connect to a database, the database tree expands to display the database hierarchy.

Disconnecting a database

You can disconnect a database in IBConsole by one of the following methods:

- Select a connected database in the Tree pane (you can also select any branch under the desired database hierarchy) and choose **Database | Disconnect** or click the Disconnect Database toolbar button
- Select a connected database in the Tree pane and choose Disconnect in the Work pane.
- Right-click a connected database in the Tree pane and choose Disconnect from the context menu.

A confirmation dialog asks you to confirm that you wish to close the connection to the selected database. Click OK if you want to disconnect the database, otherwise click Cancel.

Unregistering a database

You can unregister a disconnected database in IBConsole by one of the following methods:

- Select a database in the Tree pane (you can also select any branch under the desired database hierarchy) and choose **Database | Un-register**.
- Select a database in the Tree pane and choose Un-register in the Work pane.
- Right-click a database in the Tree pane and choose Un-register from the context menu.

A confirmation dialog asks you to confirm that you wish to un-register the database. Click OK if you want to un-register the database, otherwise click Cancel.

Note Un-registering a database automatically disconnects the current database and removes it from the Tree pane.

Connection-specific examples

Here are some examples of connecting to databases on various types of servers.

- For a Windows server, the database path name must contain the appropriate drive letter designation. For example, to connect to a local database:

```
D:\users\accting\fin\accred.gdb
```

- To connect to a database on a remote server using the TCP/IP protocol:

```
ntserver:D:\users\accting\fin\accred.gdb
```

- To connect via NetBEUI (Windows NT/2000 servers only), use UNC notation:

```
\\ntserver\D:\users\accting\fin\accred.gdb
```

- To connect via IPX/SPX (NetWare servers only) use the following notation:

```
nwserver@vol1:\accting\fin\accred.gdb
```

- For a UNIX or Linux server, you must enter the complete and absolute directory path for the database. For example:

```
server:/usr/accting/fin/accred.gdb
```

Connection troubleshooting

This section describes some troubleshooting guidelines for issues related to network configuration and client/server connections. If you are having trouble connecting client to server over a network, use the steps listed below to diagnose the cause. On Windows, you can perform some of these tests using the Communications Diagnostic dialog. See **"Communication diagnostics" on page 76** for more information.

Connection refused errors

If the client fails to reach the server host at all, or the **gds_db** service fails to answer, you might get a "connection refused" error. Below is a checklist that you can use to diagnose the source of this error.

Is there low-level network access between the client and server?

You can quickly test whether the client cannot reach the server because of a physically disconnected network or improper network software configuration, by using the **ping** command. Usage is:

```
ping servername
```

Error messages from **ping** indicate that there is a network problem. Check that the network is plugged in, that the network wires are not damaged, and that the client and server software is properly configured.

Test connectivity from the client in question to another server; if it succeeds, this could rule out improper network configuration on the client.

Test connectivity from another client to the InterBase server host; if it succeeds, this could rule out improper network configuration on the server.

Can the client resolve the server's hostname?

InterBase clients must specify the server by name, not by IP address, except in some Linux distributions. Therefore, the client must be able to resolve the server's hostname. For TCP/IP, this is done either by maintaining a **hosts** file on the client with the mappings of hostnames to IP addresses, or by the client querying a DNS server or WINS server to resolve this mapping. Make sure the name server has a correct entry for the server host in question.

Is the server behind a firewall?

If the database server is behind a software or hardware firewall, all network traffic could be restricted and the client might not be able to reach the server at all. Some firewalls permit or restrict traffic based on the port to which the client attempts to connect. Because of this, it is not conclusive whether a given service can reach the server. Neither is it an indication of connectivity if the client can resolve the IP address; that merely indicates that the client can reach a name server that resolves the InterBase server host's name.

If the client is separated from the server by a firewall, the client cannot connect.

Are the client and server on different subnets?

NetBEUI cannot route network traffic between subnets. Other protocols can also be configured to restrict traffic between subnets. If the client and server are on a complex network with multiple subnets, ask your network administrator if the network configuration allows you to route network traffic between the client and server in question using a given protocol.

Can you connect to a database locally?

To confirm that the **ibserver** process is running on the server and able to attach to your database, try a local database connection:

1. Log in to the console of the database server host, and run an application such as command-line **isql**.

2. Attempt to connect to a database without specifying a hostname: list just the path.

The Communications Diagnostic dialog also has a local database attachment test. See **"DB Connection tab" on page 76** for details.

Note Local connection mode is not available on UNIX servers or NetWare servers.

Can you connect to a database loopback?

You can simulate a client/server connection and test the server's configuration without the additional variable of the client configuration and intervening network by connecting in a *loopback* mode.

1. Log in to the console of the database server host and run an application such as command-line **isql** or InterBase IBConsole ISQL.

2. Attempt to connect to the database using a remote connection specification, even though the server named is also the client host.

Whether this test fails or succeeds, it helps to narrow the focus of further diagnostic tests. If it fails, you can infer that the server's configuration is at fault. If it succeeds, you can infer that the server is not at fault and you can concentrate further tests on the client.

Note Loopback tests cannot be performed when using NetWare, because client applications run only on a remote client, not on the NetWare server.

Is the server listening on the InterBase port?

If the **ibserver** process on the server has not started, there is no answer to attempts to connect to the **gds_db** service (port 3050).

Start the **ibserver** process on the server. Use **ibmgr -start** on UNIX, or the InterBase Manager on Windows. See **Chapter 3, "Server Configuration."**

Is the services file configured on client and server?

The **services** file must have correct entries to indicate the port number associated with the named service **gds_db**. This configuration must be accessible on the client as well as the server.

```
gds_db        3050/tcp         # InterBase Server
```

On Windows NT/2000, this file is in **C:\windows\system32\drivers\etc\services**.
On Windows 98/ME, this file is in **C:\windows\services**.
On UNIX, this file is in **/etc/services**.

In a UNIX environment with NIS, the NIS server can be configured to supply the **services** file to all NIS clients on UNIX workstations.

Is the UNIX inetd daemon configured for InterBase Classic architecture?

When running a version of InterBase that has the Superserver architecture (for instance, InterBase 6.0 for Solaris), you should check the **/etc/inetd.conf** file to make sure that the **inetd** daemon is not configured to listen on the **gds_db** service (port 3050). In InterBase SuperServer, the **ibserver** process takes over the task of listening on the port, and if both **inetd** and **ibserver** attempt to listen, then there is a conflict and the result is that neither can successfully accept connection requests.

Make the following change:

- Use a text editor to remove the line in **/etc/inetd.conf** that mentions the **gds_db** service
- Restart **inetd** by sending it a HUP signal

The installation script in InterBase for UNIX is supposed to perform this task, but if something goes wrong, or the **/etc/inetd.conf** file is restored to its configuration for InterBase Classic, you need to correct the configuration.

For InterBase versions that run in Classic mode (for instance, SCO OpenServer and Linux), check to make sure the **/etc/inetd.conf** file does have an entry for **gds_db**, and restart **inetd** with `kill -HUP` to make sure **inetd** is using the current configuration in **/etc/inetd.conf**.

Connection rejected errors

If the client reaches the server host and the **gds_db** service answers but you still cannot attach to a database, it can result in a “connection rejected” error. Below is a checklist that you can use to diagnose the source of this error.

Did you get the correct path to the database?

Verify that you supplied the correct path to the database file. Keep in mind:

- On NT/2000, you must supply the drive letter with the path.
- On UNIX, paths are case-sensitive.
- Slash (“/”) vs. backslash (“\”) does not matter, unless you need to use double-backslashes in string literals in C or C++ code.

Is UNIX host equivalence established?

To use the UNIX user-equivalence feature, there must be a *trusted host* relationship between the client and the server. See **“Users on UNIX” on page 85**.

Is the database on a networked filesystem?

A database file must not reside on an NFS filesystem or a mapped drive. When the **ibserver** process finds such a case, it either denies the connection or passes the connection request on to the InterBase service running on the file server. See **"Networked filesystems" on page 104** for more details.

To correct this situation, move your database to a filesystem on a hard disk that is physically local to the database server.

Are the user and password valid?

The client application must use a valid user and password combination that matches an entry in **isc4.gdb**. Make sure you are using a valid user and password for that server.

Does the server have permissions on the database file?

The **ibserver** process must have permission to read and write the database file at the operating system level. Check the permissions on the database file, and the uid of the **ibserver** process. (On UNIX, you have the option of running **ibserver** as user *interbase*, a non-superuser uid.)

The **isc4.gdb** database that contains users and passwords must also be writable by the **ibserver** process.

Does the server have permissions to create files in the InterBase install directory?

The **ibserver** process must have write permission in the InterBase directory (by default, **/usr/interbase** on UNIX, **C:\Program Files\Borland\InterBase** on Windows). The server process must be able to write to, and perhaps create, the **interbase.log** file and other temporary files.

Disabling automatic Internet dialup

Microsoft Windows operating systems offer a networking feature that is convenient for users who use a modem to connect to the Internet: any TCP/IP request that occurs on the system activates an automatic modem dialing program. This is helpful for users who want to connect quickly as they launch a web browser or email client application.

This convenience feature is unnecessary on systems that use a client/server application to access an InterBase server on a local network. The TCP/IP service request that the client invokes triggers the Windows automatic modem dialer. This interferes with quick network connections from client to server.

This section describes several methods to suppress the automatic modem dial feature of Windows operating systems. No more than one of these methods should be necessary to accomplish the networking configuration you need.

▸ *Reorder network adapter bindings*

You probably have a dialup adapter and an ethernet adapter for your local network. On Windows NT/2000, you can reverse the bindings order for your two adapters to force the ethernet adapter service the TCP/IP request before the dialup adapter tries. You can do this in **Control Panel | Networking | Bindings | All Adapters | Move Down**.

The local ethernet adapter satisfies TCP/IP requests it can, and those requests that can't be done locally—such as Internet requests—are passed on to the next adapter in the list, the dialup adapter.

▸ *Internet Explorer*

If you have Microsoft Internet Explorer installed, you have an item on the Control Panel that allows you to disable the autodial feature of the dialup network driver. If you don't have Internet Explorer, you won't have any control over this feature.

Run **Control Panel | Internet | Connection** tab. Check “Connect to the Internet using a local area network.”

You can also change this in some versions of Internet Explorer. Use the menu **View | Internet Options | Connection** and check “Connect to the Internet using a local area network.”

After making this change, you must invoke your modem dialer manually every time you want to use the Internet.

▸ *Disabling autodial in the registry*

Perform the following:

1. Start the registry editor with **regedit.exe**
2. Move to the registry key `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Internet Settings: EnableAutoDial`
3. Change the value from 0 to 1

▸ *Disabling RAS autodial*

The easiest way to do this is to disable the RAS AutoDial service:

1. Start the services control panel applet **Control Panel | Services**
2. Scroll down to “Remote Access AutoDial Manager” and select it
3. Click **Startup** and change the startup to Manual; click **OK**

4. If you want to stop it now click **Stop**
5. Click **Close**

To re-enable the RAS autodial service, repeat the above but change the startup to Automatic.

▶ *Preventing RAS from dialing out for local network activity*

Perform the following if you are using Windows NT RAS:

1. Start the registry editor, with **regedit.exe**
2. Move to the registry key
   `HKEY_CURRENT_USER\Software\Microsoft\RAS Autodial\`
   `Addresses`

   A better way to view these is to type **rasautou -s** from the command prompt
3. In the subkeys look for the local address and name; select the key and select **Delete** from the **Edit** menu
4. Close the registry editor

You might also wish to add addresses to the disabled list:

1. Start the registry editor with **regedt32.exe**, not **regedit.exe**
2. Move to the registry key
   `HKEY_CURRENT_USER\Software\Microsoft\RAS Autodial\Control`
3. Double click Disabled Addresses and add the address on a new line; click **OK** when you are finished
4. Close the registry editor

You must reboot the machine in both of the above cases.

Other errors

Unknown Win32 error 10061

This error is often associated with a missing server-access license for the InterBase software on the server host. Make sure you have licensed InterBase server to allow clients to connect from the network. See the Installation chapter of *Getting Started* for information about licensing InterBase at install time and “Using the Install and Licensing APIs” in the *Developer’s Guide* for information about using the Licensing API to license InterBase as part of an application install.

Unable to complete network request to host

This error occurs in cases when the InterBase client cannot establish a network connection to the server host. This can occur for a variety of reasons. Below is a list of common causes:

- The BDE Administrator requires that you specify the InterBase connect string in the `SERVER NAME` alias property. You must use this property and must not use the `PATH` alias property, or else you receive the network error message.
- The InterBase client attempts to translate the server portion of your connect string to an IP address, by calling `gethostbyname()`. If you supplied an IP address, `gethostbyname()` is likely to fail to resolve it. Some modern TCP/IP drivers—including Winsock 2 and Linux TCP/IP—do resolve strings that look like IP addresses. If you are on Windows, specify hosts by name, or else upgrade your TCP/IP driver to Winsock 2.
- The InterBase client must look up the InterBase network *service* by name. If the client doesn't find the entry for **gds_db** in the **services** file, it might fail to connect to the server, and give the network error. You can create the entry in the **services** file manually, or reinstall InterBase to perform this task.
- The server you specify must be running on the network that you use. If the hostname corresponds to a host that is inaccessible because of network interruption, or the host is not running, then the connection request fails with the network error.
- The syntax of the InterBase connect string determines the network protocol the client uses to connect to the server host (see **"Connection-specific examples" on page 68**). Different server platforms support different subsets of network protocols. If your server does not support the protocol indicated by your connect string, the connection attempt fails with the network error. For example, the NetBEUI connection syntax (**\\server\C:\path\database.gdb**) works only if your server is a windows NT or Windows 2000 server. The syntax does not work if your server is running UNIX, Linux, or NetWare.
- A network connection request succeeds only if the InterBase server is installed and active on the server host, and that the InterBase server is licensed to receive remote connection requests. If there is no process listening for connection requests, the client's connection requests with the network error. You should check that the InterBase server is installed on the server, that it is running, and that the license includes the Server capability.
- The InterBase 32-bit client library prior to version 5.1 does not support IPX/SPX network protocol. If you attempt to use IPX/SPX by specifying the database connect string in the form **server@volume:/path/database.gdb**, it fails with the network error message. Upgrade your InterBase client to version 5.1 or later.

Communication diagnostics

Network configuration of a client/server system involves several different software and hardware layers and proper configuration of each of these layers. When one or more layers are misconfigured, it is not always evident where the problem lies. InterBase Communication diagnostics helps to identify the source of the problem by testing each layer progressively for existing or potential network problems.

You can access the Communication Diagnostics dialog by one of the following methods:

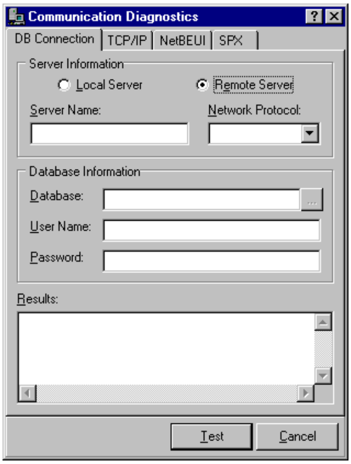
- Select a disconnected server in the Tree pane. Choose **Server | Diagnose Connection.**
- Right-click InterBase Servers or any disconnected server in the Tree pane and choose Diagnose Connection from the context menu.
- Select a disconnected server from the Tree pane and choose Diagnose Connection in the Work pane.

There are four types of diagnostics that you can perform. The Communications Diagnostics dialog has separate tabs for each diagnostic type.

DB Connection tab

This test lets you connect to an InterBase database using the InterBase client libraries. It is the most basic test of InterBase operation and is generally used only after confirmation that the underlying network is working correctly.

FIGURE 4.5 Communications dialog - DB Connection



Server Information: ○ Local Server ● Remote Server

▸ *To run a DB Connection test:*

1. Select either the Local Server option or the Remote Server option.
2. If you choose Local Server, the Server Name and Network Protocol information is not required. These text fields are disabled. You can proceed to step 5.
3. If you choose Remote Server, type the name of the server in the Server Name text field.

   Note The InterBase server name is the name of the database server machine. There is not a specific name for the InterBase server process itself. For example, if the server is running on the NT server "**venus**", you enter this name in the Server Name text field.
4. If you choose Remote Server, select a network protocol from the drop down list: either TCP/IP, NetBEUI, or SPX. Protocols are valid only when they are supported by both the client and the server.
5. Enter the database filename, including the path where file is located, in the Database text field. If you selected the Local Server option in step 1 you can also click the browse button … to locate the file you want. If you selected the Remote Server option, however the browse button is disabled.
6. Type the username and password for the database in the corresponding User Name and Password text fields.
7. Click Test to display the results of the connectivity test in the Results text area.

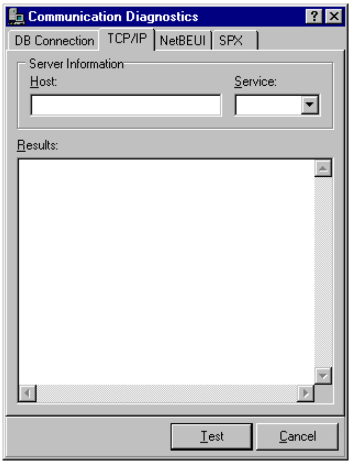
▸ *Sample output (local connection)*

```
Attempting to attach to:
   C:\Program Files\InterBase Corp\InterBase\
      Examples\employee.gdb

   Attaching ...Passed!
   Detaching ...Passed!

InterBase Communication Test Passed!
```

TCP/IP tab

Use this property sheet to test Winsock TCP/IP connectivity.

FIGURE 4.6 Communications dialog - TCP/IP



To run a winsock TCP/IP connectivity test:

1. Enter either a network host name or IP address in the Host text field.
2. Select a service name or number from the dropdown Service list. Possible service selections are: 21, Ping, 3050, ftp, **gds_db**.

   Select Ping from the Service dropdown list to display a summary of round-trip times and packet loss statistics.
3. Click Test to display the results of the connectivity test in the Results text area.

Sample results (ftp):

```
Initialized Winsock.

Attempting connection to DBSERVE.
Socket for connection obtained.

Found service ‘FTP’ at port ‘21’.
Connection established to host ‘DBSERVE’ on port 21.

TCP/IP Communication Test Passed!
```

Sample results (ping):

```
Pinging DBSERVE [200.34.4.5] with 32 bytes of data.

Reply from 200.34.4.5: bytes=32 time=1ms TTL=128
Reply from 200.34.4.5: bytes=32 time=1ms TTL=128
Reply from 200.34.4.5: bytes=32 time=1ms TTL=128
Reply from 200.34.4.5: bytes=32 time=0ms TTL=128

Ping statistics for 200.34.4.5:
    Packets: Send = 4, Received = 4, Lost = 0 (0%),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms
```

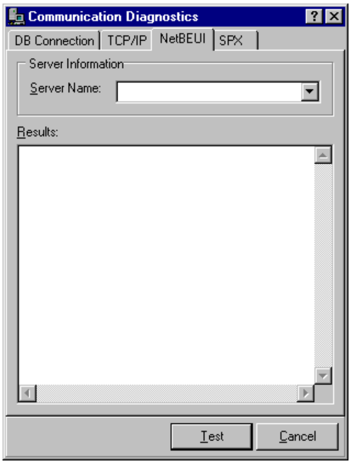
If the error message is...	Then check...
Failed to find named port	• Your **services** file to be sure there is an entry for **gds_db** in the form: **gds_db 3050/tcp**
Failed to connect to host	• Hostname, port 3050 • The InterBase Server to make sure it is installed properly, is running, and is configured for TCP/IP
Failed to resolve hostname	• Hostname • Your **hosts** file or DNS to be sure it has an entry for the server • That you used a hostname and not an IP address
Unavailable database	• Whether the InterBase server is running; the server must be running before attempting a database connection

TABLE 4.2 Using Communication Diagnostics to diagnose connection problems

NetBEUI tab

Use this property sheet to test NetBEUI connectivity between the client and the server.

FIGURE 4.7 Communications dialog - NetBEUI



To run a NetBEUI connectivity test:

1. Select a Windows NT or Windows 2000 server on which InterBase has been installed from the Server Name drop down list. If the desired server does not exist in this list, you can type the server name in the edit portion of the drop down list.
2. Click Test to display the results of the connectivity test in the Results text area.

Sample output (NetBEUI connection):

```
Attempting to attach to DBSERVE using
the following named pipe:
  \\dbserve\pipe\interbas\server\qds.db.

NetBEUI Communication Test Passed!
```

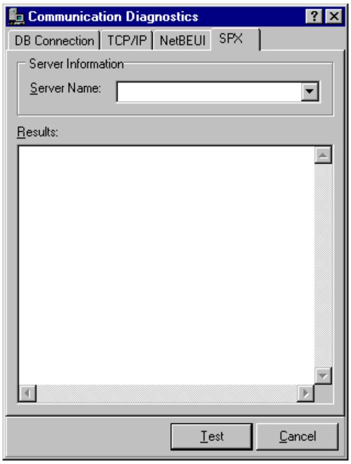
Note NetBEUI is supported on all Windows clients, but only Windows NT and Windows 2000 support NetBEUI as a server.

The connection may fail if a Microsoft Windows network is not the default network for the client. You should also be logged into the MS Windows network with a valid NT/2000 user name and password.

SPX tab

Use this property sheet to test SPX connectivity between the client and the server.

FIGURE 4.8 Communications dialog - SPX



To run an SPX connectivity test:

1. Select a name of a Windows NT/2000 server, on which InterBase has been installed, from the Server Name drop down list. If the desired server does not exist in this list, you can also type the server name in the edit portion of the drop down list.
2. Click Test to display the results of the connectivity test in the Results text area.

Sample output (SPX connection):

```
Attempting to attach to DBSERVE using SPX.

Attached successfully to DBSERVE
using the SPX protocol.

SPX Communication Test Passed!
```

Note SPX is supported on all Windows clients, but only Novell supports SPX as a server.

CHAPTER

5

Database Security

InterBase provides several methods for configuring and enforcing security by controlling how a database is accessed and used. Server security enables you to:

- Add a user to the security database
- Delete a user from the security database
- Modify user information in the security database
- Display a list of users in the security database

This chapter gives an overview of these options. The user administration tools are covered here, but SQL statements for configuring privileges are in other InterBase books; these passages are referenced where appropriate.

Security model

Security for InterBase relies on a central security database for each server host. This database, **isc4.gdb**, contains a record for each legitimate user who has permission to connect to databases and InterBase services on that host. Each record includes the user login name and the associated encrypted password. The entries in this security database apply to all databases on that server host.

The username is significant to 31 characters and is not case sensitive. Password is significant to eight characters and is case sensitive.

Before performing any database administration tasks, you must first log in to a server. Once you log in to a server, you can then connect to databases residing on the server.

All users must enter their username and password to log in to a server. The password is encrypted for transmission over the network. The username and password are verified against records in the security database. If a matching record is found, the login succeeds.

The SYSDBA user

Every InterBase server has a SYSDBA user, with default password *masterkey*. SYSDBA is a special user account that can bypass normal SQL security and perform tasks such as database backups and shutdowns.

Initially, SYSDBA is the only authorized user on a server; the SYSDBA must authorize all other users on the server. Only the SYSDBA user can update the security database to add, delete, or modify user configurations. SYSDBA can use either **gsec** or IBConsole to authorize a new user by assigning a username and password in the security database.

IMPORTANT We *strongly* recommend you change the password for SYSDBA as soon as possible after installing InterBase. If you do not alter the SYSDBA password, unauthorized users have easy access and none of your databases are secure.

Other users

The SYSDBA account can create other users on a per-server basis. Use **gsec** or IBConsole to create, modify, or remove users from the **isc4.gdb** security database. These users are authorized to connect to any database on that database server host. It is a common design strategy to create a distinct InterBase user for each person who uses the databases on your server. However, other strategies are also legitimate. For example:

- Create one InterBase user for an entire group of people to use, in order to simplify password administration. For example, a user FINANCE could satisfy the access needs for any and all staff in a financial analysis team. This team only needs to remember one password between them.
- Create one InterBase user for a group of people to use, as warranted by requirements of distinct privilege configurations. For example, if Erin and Manuel have identical access to the data within a database, they could use the same InterBase user account.

Users on UNIX

If both the client and the server are running UNIX, you can allow UNIX usernames access to databases by configuring the server host to treat the client host as a *trusted host*.

To establish a trusted host relationship between two hosts, add an entry in **/etc/hosts.equiv** or **/etc/gds_hosts.equiv** on the server. The former file establishes trusted host status for any service (for example, **rlogin**, **rsh**, and **rcp**); the latter file establishes trusted host status for InterBase client/server connections only. The format of entries in both files is identical; see your operating system documentation on **hosts.equiv** for details.

The login of the client user must exist on the server. In addition to the **hosts.equiv** method of establishing a trusted host, the you can also use the **.rhosts** file in the home directory of the account on the server that matches the account on the client.

The InterBase client library defaults to using the current client's UNIX login as the InterBase login only when the client specifies no username through any of the following methods:

- Database parameter buffer (*dpb*) parameters—see the *API Guide*
- Command-line options—for example, **-user** options of **isql** or another utility
- Environment variables—see **"ISC_USER and ISC_PASSWORD" on page 51**

Notes

- This feature is not implemented in InterBase 6 on a Windows NT or Windows 2000 server, because Windows does not implement a trusted host mechanism as UNIX does.
- Windows clients cannot be treated as trusted hosts by UNIX servers.

Security database isc4.gdb

Every user of an InterBase server requires an entry in the **isc4.gdb** security database. The **gsec** security utility lets you display, add, modify, or delete information in **isc4.gdb**. IBConsole provides a graphical interface for the same functionality. The following table describes the contents of **isc4.gdb**:

Column	Required?	Description
User name	Yes	The name that the user supplies when logging in; maximum length is 31 characters
Password	Yes	The user's password • Case sensitive • Only the first eight characters are significant • Maximum length: 32 characters.
UID	No	An integer that specifies a user ID
GID	No	An integer that specifies a group ID
Full name	No	User's real name (as opposed to login name)

TABLE 5.1 Format of the **isc4.gdb** security database

SQL privileges

Connecting to a database does not automatically include privileges to modify or even view data stored within that database. Privileges must be granted explicitly; users cannot access any database objects until they have been granted privileges. Privileges granted to PUBLIC apply to all users.

For full description of syntax of SQL privileges, see entries for GRANT and ROLE in the *Language Reference* and *Data Definition Guide*.

Groups of users

InterBase implements features for assigning SQL privileges to groups of users. SQL roles are implemented on a per-database basis. UNIX groups are implemented on a server-wide basis, using the UNIX group mechanism; this feature is not available on Windows or NetWare platforms.

ANSI SQL 3 roles

InterBase supports SQL group-level security as described in the *ISO-ANSI Working Draft for Database Language*. For syntax of SQL ROLEs, see the *Language Reference* and *Data Definition Guide*.

Implementing roles is a four-step process.

1. Declare the role with CREATE ROLE.

   ```
   CREATE ROLE sales;
   ```

2. Assign privileges on specific tables and columns to the role using the GRANT statement.

   ```
   GRANT UPDATE ON table1 TO sales;
   ```

3. Grant the role to users, again with the GRANT statement.

   ```
   GRANT sales TO user1, user2, user3;
   ```

4. Finally, to acquire the privileges assigned to a role, users must specify the role when connecting to a database.

   ```
   CONNECT 'foo.gdb' USER 'user1' PASSWORD 'peanuts' ROLE sales;
   ```

User1 now has update privileges on TABLE1 for the duration of the connection.

A user can belong to only one role per connection to the database and cannot change role while connected. To change role, the user must disconnect and reconnect, specifying a different role name.

You can adopt a role when connecting to a database by any one of the following means:

- **To specify a role when attaching to a database through IBConsole ISQL**, display the Database Connect dialog and type a rolename in the Role field.
- **To specify a role programmatically upon connection using the InterBase API**, use the dpb parameter *isc_dpb_sql_role_name*. See chapter 4 of the *API Guide*.
- **To specify a role for a connection made by an embedded SQL application or isql session**, use the ROLE *rolename* clause of the CONNECT statement. See the statement reference for CONNECT in the *Language Reference*.

Note Applications using BDE version 5.02 or later, including Delphi, JBuilder, and C++Builder, have a property by which they can specify a role name. Also, the ODBC driver that currently ships with InterBase also recognizes roles.

UNIX groups

Operating system-level groups are implicit in InterBase security on UNIX, similarly to the way UNIX users automatically supplement the users in **isc4.gdb**. For full description of usage and syntax of using UNIX groups with InterBase security, see the *Language Reference* and *Data Definition Guide*.

Note Integration of UNIX groups with database security is not an SQL standard feature.

Other security measures

InterBase provides some restrictions on the use of InterBase tools in order to increase security. In addition, there are things that you can do to protect your databases from security breaches. This section describes these options.

Restriction on using InterBase tools

As a security measure, InterBase requires that only the owner of a database or SYSDBA can execute **gbak**, **gstat**, and **gfix**.

- Only the database owner or SYSDBA can use **gbak** to back up a database. Anyone can restore a database, because there is no concept of an InterBase user for a backup file. However, only the owner or SYSDBA can restore a database over an existing database. For security purposes, make sure that your backup files are stored in a secure location. This prevents unauthorized persons from restoring databases and gaining access to them.
- On UNIX platforms, there is a further constraint on **gstat**: to run **gstat**, you must have system-level read access to the database file. To access the database with **gstat**, you must either be logged into the account running the InterBase server ("interbase" or "root") or someone must change the permissions on the database file to include read permission for your Group.

Protecting your databases

You can take several steps to increase the security of your databases and other files on your system:

- UNIX and Linux systems: Before starting the InterBase server, log in as user "interbase" (or "interbas", if user names longer than eight characters are not allowed), rather than "root" (only these users can start the server). This restricts the ability of other users to accidentally or intentionally access or overwrite sensitive files such as the password file. Start the InterBase server while you are logged on as user "interbase".
- Windows NT/2000 systems: When the InterBase server is run as a service, you can protect a database against unauthorized access from outside InterBase (such as by a **copy** command), by making the database files readable only by the system account, under which services run. However, if you make the database readable only by the system account, remote access to the database must be by TCP/IP, not by NetBEUI.

- Windows 98/ME systems: When security is an issue, it is inappropriate to run an InterBase server on Windows 98 at all. Due to the lack of a file-permission system, an attacker can readily duplicate the database with the **copy** command or with a point-and-click equivalent. While sometimes a stolen copy of a database will be corrupt and unusable, it might turn out to be adequate, especially if update activity on the database is relatively infrequent, or if the attacker can try copying repeatedly.
- Because anyone can restore a backed up database, it is wise to keep your backup files in a directory with restricted access. (On a Windows 98/ME platform, you can either move backup files to physical media such as tape or high-density removable drives and store these securely, or move the backup files to a system that restricts directory access). On UNIX, only the backup file itself, not the directory in which it resides, needs to have permissions restricted to prevent reading by unauthorized persons.

  For example, if all of the following are true:

  · the backup file has permission 600 (rw-------) or 640 (rw-r-----)

  · only trusted persons belong to the groups

  · the directory has permission rwxr-xr-x

  then persons other than the responsible owner and group can see that the backup file is there, but they cannot get at it. If the user or backup script issues the command “umask 077” (or 027, as appropriate) before running **gbak**, unauthorized persons will not be able to access the backup file, no matter what the permissions on the directory. (Of course the directory should not be writable by “other”; that would permit other persons to delete the backup file.)

User administration with IBConsole

User administration is accomplished through the User Information dialog where you are able to add, modify, view and delete users. User administration can only be performed after logging in to the server.

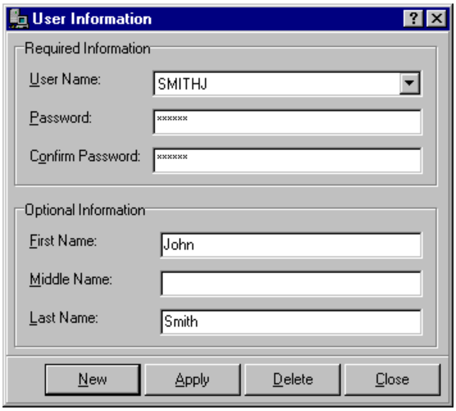
Displaying the User Information dialog

You can use any of the following methods to access the User Information dialog:

- Select a logged in server or any branch under the server hierarchy from the list of registered servers in the Tree pane; choose **Server | User Security**.

- Select a logged in server from the list of registered servers in the Tree pane. Click User Security in the Work pane or right-click the selected server and choose User Security from the context menu.
- Select Users under the desired server in the Tree pane to display a list of valid users in the Work pane. Double-click a username in the Work pane.

FIGURE 5.1 User information dialog



User Information

Required Information

User Name: SMITHJ

Password: ******

Confirm Password: ******

Optional Information

First Name: John

Middle Name:

Last Name: Smith

Adding a user

Use the User Information dialog to add new users. To access this dialog follow one of the methods described in **"Displaying the User Information dialog" on page 89**.

To add a new user:

1. Click New. The New and Delete buttons are disabled and the Close button changes to a Cancel button.
2. Type the new username in the User Name text field.
3. Type the user's password in both the Password and the Confirm Password text fields.
4. Add any desired optional information in the corresponding text fields. Each of the optional text fields can be up to 32 characters.
5. Click Apply to add the new user to the security database or click Cancel to abandon your changes.

Note Usernames can be up to 31 characters long and are *not* case sensitive. Passwords can be up to 32 characters long and *are* case-sensitive. Only the first eight characters of the password are significant. InterBase does not allow you to create usernames or passwords containing spaces.

Modifying user configurations

Use the User Information dialog to modify user configurations. To access this dialog follow one of the methods described in **"Displaying the User Information dialog" on page 89**.

To modify a user's details:

1. From the User Name drop down list, select the user whose configuration you wish to modify. The user's details display. You can also type the first letter of the desired username in the User Name drop down list to quickly scroll to usernames beginning with that letter. By repeatedly typing that same letter, you can scroll through all usernames that begin with that letter.
2. Change any of the text fields except the User Name. If you change the password, you must enter the same password in the Password text field and the Confirm Password text field.
3. CSlick the Apply button to save your changes.

Note You may not modify a username. The only way to change a username is to delete the user and then add a user with the new name.

Deleting a user

Use the User Information dialog to removd users from the security database. To access this dialog follow one of the methods described in **"Displaying the User Information dialog" on page 89**.

To remove a user account:

1. Select the user you wish to delete from the User Name drop down list. You can also type the first letter of the desired username in the User Name drop down list to quickly scroll to usernames beginning with that letter. By repeatedly typing that same letter, you can scroll through all usernames that begin with that letter.
2. Click Delete. A confirmation dialog inquires, “Do you wish to delete user *username*?” If you choose OK, the user is removed and is no longer authorized to access databases on the current server.

IMPORTANT Although it is possible for the SYSDBA to delete the SYSDBA user, it is strongly not recommended because it will no longer be possible to add new users or modify existing user configurations If you do delete the SYSDBA user, you must reinstall InterBase to restore the **isc4.gdb** security database.

User administration with the InterBase API

The InterBase API includes three functions that permit authors of InterBase applications to add, delete, and modify users programmatically using three API functions: These functions are *isc_add_user*( ), *isc_delete_user*( ), and *isc_modifiy_user*( ). These functions are deprecated in InterBase 6 and later, however, because they are replaced by functions in the InterBase Services API.

The InterBase Services API provides a much broader and more robust set of tools for programmatically managing users in the security database. See Chapter 12: "Working with Services" in the *API Guide* for details and examples of using the Services API functions.

For programmers using Delphi and C++ Builder, the IBX components for InterBase 6 and higher include components for managing users. For more information on using the IBX components, refer to the *Developer’s Guide*.

Note Delphi 5 ships with an older version of IBX that does not include Services API components. Install the newer version of IBX that is included with InterBase 6 and later.

Using gsec to manage security

The InterBase command-line security utility is **gsec**. This utility is used in conjunction with the security database **isc4.gdb**, to specify user names and passwords for an InterBase server. This tool duplicates the functionality of **Server | User Security** in IBConsole for Windows.

The security database, **isc4.gdb**, resides in the InterBase install directory. To connect to a database on the server, users must specify a user name and password, which are verified against information stored in **isc4.gdb**. If a matching row is found, the connection succeeds.

IMPORTANT Only the SYSDBA can run **gsec**. To do this, use one of the following methods:

- Invoke the command as:

```
gsec -user sysdba -password masterkey
```

- Define the ISC_USER and ISC_PASSWORD environment variables for SYSDBA before you invoke the command.
- Run **gsec** when you are logged in as root on UNIX or Administrator on Windows NT/2000.

To use **gsec** interactively, type **gsec** at the command prompt. The prompt changes to `GSEC>`, indicating that you are in interactive mode. To quit an interactive session, type `QUIT`.

Running gsec remotely

You can use **gsec** on a client host to administer users in a security database on a remote server. Use the **-database** option with a remote database specification to connect to a remote **isc4.gdb**. For example:

```
gsec -database jupiter:/usr/interbase/isc4.gdb
```

Using gsec commands

The following table summarizes **gsec** commands. The initial part of each command is required. The part in brackets is optional.

Command	Description
di[**splay**]	Displays all rows of **isc4.gdb**
di[**splay**] *name*	Displays information only for user *name*
a[**dd**] *name* **-pw password** [*option argument*] [*option argument* **...**]	Adds user *name* to **isc4.gdb** with password *string*. Each *option* and corresponding *argument* specifies other data associated with the user, as shown in **Table 5.3, "gsec options"**
mo[**dify**] *name* [*options*]	Like add, except that *name* already exists in **isc4.gdb**
de[**lete**] *name*	Deletes user *name* from **isc4.gdb**
h[**elp**] or **?**	Displays **gsec** commands and syntax
q[**uit**]	Quits the interactive session

TABLE 5.2 Summary of **gsec** commands

Displaying the security database

To see the contents of **isc4.gdb**, enter the DISPLAY command at the GSEC> prompt. All the rows in the security database are displayed:

```
GSEC> display
user nameuid     gid full name
----------------------------------------------
JOHN      123    345 John Doe
JANE      124    345 Jane Doe
RICH      125    345 Richard Roe
```

Note that passwords are never displayed.

Adding entries to the security database

To add users to the security database, use the **add** command:

```
a[dd] name -pw password [options]
```

followed by a user name, the **-pw** option followed by a password, and any other options, as shown in the following table. The password is case sensitive. None of the other parameters are case sensitive.

For each option, the initial letter or letters are required and optional parts are enclosed in brackets. Each option must be followed by a corresponding argument, a string that specifies the data to be entered into the specified column in **isc4.gdb**.

Option	Meaning
-password or **-pa** *string*	Password of user who is performing the change
-user *string*	User who is performing the change
-pw *string*	Target user password
-uid *integer*	Target user ID
-gid *integer*	Group ID for target user
-fname *string*	First Name for target user
-mname *string*	Middle Name for target user
-lname *string*	Last Name for target user

TABLE 5.3 **gsec** options

Note The **-pa** switch specifies the root or the SYSDBA account password; **-pw** specifies the password for the user being added or modified.

For example, to add user "jones" and assign the password "welcome", enter:

```
GSEC> add jones -pw welcome
```

Use **display** to verify the entry. An unassigned UID or GID defaults to 0:

```
GSEC> display
user name     uid     gid     full name
----------------------------------------------
JONES         0       0
```

For example, to add authorization for a user named Cindi Brown with user name "cbrown" and password "coffee2go", use the following **gsec** command:

```
GSEC> add cbrown -pw coffee2go -fname cindi -lname brown
```

To verify the new entry, display **isc4.gdb**:

```
GSEC> display
user name     uid     gid     full name
----------------------------------------------
JONES         0       0
CBROWN        0       0       CINDI  BROWN
```

gsec stores the user name in uppercase regardless of how it is entered.

▸ *Modifying the security database*

To change existing entries in the security database, use the **modify** command. Supply the user name for the entry to change, followed by the option indicating the items to change and the corresponding values to which to change them.

For example, to set the user ID of user "cbrown" to 8 and change the first name to "Cindy", enter the following commands:

```
GSEC> modify cbrown –uid 8 –fname cindy
```

To verify the changed line, use **display** followed by the user name:

```
GSEC> display cbrown
user name        uid    gid    full name
---------------------------------------------
CBROWN           8      0      CINDY BROWN
```

Note To modify a user name, first delete the entry in **isc4.gdb**, then enter the new user name and re-enter the other information.

▸ *Deleting entries from the security database*

To delete a user's entry in **isc4.gdb**, use **delete** and specify the user name:

```
GSEC> delete cbrown
```

You can confirm that the entry has been deleted with the **display** command.

Using gsec from a Windows command prompt

To use **gsec** from the Windows NT or Windows 2000 command prompt, precede each command with **gsec** and prefix each **gsec** command with a hyphen (-). For example, to add user "aladdin" and assign the password, "sesame", enter the following at the command line:

```
C:> gsec –add aladdin –pw sesame
```

To display the contents of **isc4.gdb**, enter:

```
C:> gsec –display
```

gsec error messages

Error Message	Causes and Suggested Actions to Take
Add record error	The **add** command either specified an existing user, used invalid syntax, or was issued without appropriate privilege to run **gsec**. Change the user name or use **modify** on the existing user.
<string> already specified	During an **add** or **modify**, you specified data for the same column more than once. Retype the command.
Ambiguous switch specified	A command did not uniquely specify a valid operation.
Delete record error	The **delete** command was not allowed. Check that you have appropriate privilege to use **gsec**.
Error in switch specifications	This message accompanies other error messages and indicates that invalid syntax was used. Check other error messages for the cause.
Find/delete record error	Either the **delete** command could not find a specified user, or you do not have appropriate privilege to use **gsec**.
Find/display record error	Either the **display** command could not find a specified user, or you do not have appropriate privilege to use **gsec**.
Find/modify record error	Either the **modify** command could not find a specified user, or you do not have appropriate privilege to use **gsec**.
Incompatible switches specified	Correct the syntax and try again.
Invalid parameter, no switch defined	You specified a value without a preceding argument.
Invalid switch specified	You specified an unrecognized option. Fix it and try again.
Modify record error	Invalid syntax for **modify** command. Fix it and try again. Also check that you have appropriate privilege to run **gsec**.

TABLE 5.4 **gsec** security error messages

Error Message	Causes and Suggested Actions to Take
No user name specified	Specify a user name after **add**, **modify**, or **delete**.
Record not found for user: <*string*>	An entry for the specified user could not be found. Use **display** to list all users, then try again.
Unable to open database	The **isc4.gdb** security database does not exist or cannot be located by the operating system.

TABLE 5.4 **gsec** security error messages (*continued*)

CHAPTER

6

Database Configuration and Maintenance

This chapter describes configuration and maintenance issues for individual databases, including the following topics:

- **Database files**
- **On-disk structure (ODS)**
- **Read-write and read-only databases**
- **Creating databases**
- **Backup file properties**
- **Shadowing**
- **Setting database properties**
- **Sweep interval and automated housekeeping**
- **Configuring the database cache**
- **Forced writes vs. buffered writes**
- **Validation and repair**
- **Shutting down and restarting databases**

- **Limbo transactions**
- **gfix command-line tool**

Database files

InterBase database files are in many cases self-contained. All the data and indexes are maintained as data structures within one type of file. The transaction log is also kept within this file.

You can extend the functions available in InterBase database metadata by creating libraries of functions compiled in your language of choice. You can compile functions into a dynamic library (called a DLL on Windows, and a shared library on UNIX) and use them in queries, stored procedures, triggers, views, and so on.

Database file size

InterBase database file size is the product of the number of database pages times the page size. The minimum page size is 1 KB, the default page size is 4KB, and the maximum page size is 8KB. Each page can store records only from a single table. You set the database page size when you create a database by using the PAGE SIZE clause of the CREATE DATABASE statement, or its equivalent in IBConsole. You can change the page size when you restore a database using **gbak** or IBConsole.

Database files cannot exceed 4GB total size (2GB on Windows 98 and Linux). When a database file reaches 4GB, InterBase issues a warning and stops writing data to that file. Before the database reaches this point, you should add one or more secondary files using the ALTER DATABASE statement, or IBConsole, or by using **gbak** to back the database up and then restore it.

Note Using **gbak** is the *only* way to reduce the size of the primary database file. When you restore a database, you can specify multiple files without reference to the original file sizes.

Dynamic file sizing

InterBase dynamically expands the last file in a database as needed until it reaches the 4GB limit (2GB on Windows 98 and Linux). This applies to single-file database as well as to the last file of multifile databases. Specifying a LENGTH for for the last or only file in a database has no effect.

External files

InterBase permits external files to be used as *external tables*. These tables are limited in their functionality:

- From a database that is in read-write mode, you can execute only SELECT and INSERT statements on external tables. From a read-only database, you can execute only SELECT statement on external tables.
- You cannot define indexes on external tables; they are outside of the control of the multigenerational architecture.

Temporary files

InterBase dynamically creates files in the temporary file space for scratch space during sorting operations involving large amounts of data. See **"Managing temporary files" on page 52** for details on temporary file use.

File naming conventions

InterBase database files are given a file extension of **.gdb** by convention, though the software does not enforce this and you can choose to use another file extension. For purposes of this documentation, assume that **.gdb** refers to an InterBase database file type.

InterBase is available on a wide variety of platforms. In most cases users in a heterogeneous networking environment can access their InterBase database files regardless of platform differences between client and server machines if they know the target platform's file naming conventions.

Because file naming conventions differ widely from platform to platform, and because the core InterBase documentation set is the same for each of these platforms, all file names in text and in examples are restricted to a base name with a maximum of eight characters, with a maximum extension length of three characters. For example, the sample database on all servers is referred to as **employee.gdb**.

Generally, InterBase fully supports each platform's file naming conventions, including the use of node and path names. InterBase, however, recognizes two categories of file specification in commands and statements that accept more than one file name. The first file specification is called the *primary file specification*. Subsequent file specifications are called *secondary file specifications*. Some commands and statements place restrictions on using node names with secondary file specifications. In syntax statements, file specification is denoted as '*filespec*'

Primary file specifications

InterBase syntax always supports a full file specification, including optional node name and full path, for primary file specifications. For example, the syntax notation for CREATE DATABASE appears as follows:

```
CREATE {DATABASE | SCHEMA} 'filespec'
   [USER 'username' [PASSWORD 'password']]
   [PAGE_SIZE [=] int]
   [LENGTH [=] int [PAGE[S]]]
   [DEFAULT CHARACTER SET charset]
```

In this syntax, the *filespec* that follows CREATE DATABASE supports a node name and path specification, including a platform-specific drive or volume specification.

Secondary file specifications

For InterBase syntax that supports multiple file specification, such as CREATE DATABASE, all file specifications after the first one are *secondary*. Secondary file specifications cannot include a node name, but can specify a full path name.

Multifile databases

InterBase supports databases that span multiple files and multiple filesystems. You can add additional files to the database without having to take it off line.

The Database Restore task in IBConsole and in the **gbak** command-line utility permit you to create a multifile database. The only way to alter the file size allocation of an existing database is to back up and restore the database file.

Adding database files

You have the option of specifying the size of secondary files in either of two ways: specify the page on which each secondary file starts, or specify the length in database pages of each file. When you specify the size using the LENGTH keyword, do not specify the length of the final file. InterBase sizes the final file dynamically, as needed.

The following **isql** example adds files using STARTING AT syntax:

```
CONNECT ‘first.gdb’;

ALTER DATABASE
   ADD FILE 'second.gdb' STARTING AT 50000;
```

Altering database file sizes

You cannot use ALTER DATABASE to split an existing database file. For example, if your existing database is 80,000 pages long and you issue the command above, InterBase starts the new database file at page 80,001. The only way to split an existing database file into smaller files is to back it up and restore it. When you restore a database, you are free to specify secondary file sizes at will, without reference to the original file sizes.

The following **isql** example adds a file using LENGTH syntax. **second.gdb** will begin on the page following the final page of **first.gdb** and will grow to 50,000 database pages. Then InterBase begins writing to **third.gdb** and dynamically increases the size as necessary.

```
CONNECT 'first.gdb';
ALTER DATABASE ADD FILE 'second.gdb' LENGTH 50000
    ADD FILE 'third.gdb';
```

InterBase starts writing data to **third.gdb** only after **second.gdb** file fills up. In the example above, **second.gdb** is 50,000 pages long, and begins following the original file. InterBase will begin filling the **third.gdb** file after **second.gdb** reaches 50,000 pages. Database pages are 4KB each by default and have a maximum size of 8KB.

There is no guarantee that a given table resides entirely in one file or another. InterBase stores records based on available space within database files. Over time, records from a given table tend to spread over all the files in a multifile database.

Maximum number of files

InterBase allows up to 65,536 database files, including shadow files. Note that your operating system might have a much lower limit on the number of simultaneous open files than the **ibserver** process can have.

Application considerations

A multifile database is not the same thing as multiple single-file databases. The tables are all part of the same database they used to be in, but they can be stored across the multiple files. From your application's standpoint, they're all part of the same database and are accessed exactly the same way they would be in a single-file database.

Your application does not need to know about any files except the first one. Any time your database operations access/write data in the secondary files, the InterBase software takes care of it without requiring any special programming from your application. The application attaches to the database by specifying the path of the first file of the database; applications don't change.

Reorganizing file allocation

You can change the sizes of the files of a multifile database when using **gbak** to restore a database. If you need to move a multi-file database to a different disk or directory, use **gbak** to back up the database, then specify the new locations of all secondary files as you restore the database. See **"gbak command-line tool" on page 157**.

Tip Any database in a production environment should include a definition for at least one secondary file, even if the current size of the database does not warrant a multifile database. Data tends to accumulate without bounds, and some day in the future your database might exceed your filesystem size, or the operating system's maximum file size. By defining a secondary file, you specify what action InterBase takes when the database grows beyond these limits. This means that the database administrator is freed from monitoring the database as it approaches the file size limit.

Networked filesystems

An InterBase database must reside on a disk local to the server software that accesses it. The database file (including any secondary files and shadow files) cannot reside on networked or remote filesystems (called mapped drives on Windows and NFS filesystems on UNIX). External tables and UDF libraries can reside on networked filesystems, but this practice is not recommended because networked filesystems can suffer from intermittent availability.

On UNIX, the InterBase software detects that a database file is located on an NFS filesystem. In this case, it invokes the remote access method to contact an InterBase server process running on the host that exported the filesystem. If there is no InterBase server software running on that node, any connection to the database fails.

On-disk structure (ODS)

Each release of InterBase has characteristic features in its internal file format. To distinguish between the file formats, InterBase records an on-disk structure (ODS) number in the **.gdb** file. In general, major ODS versions (those incrementing the number to the left of the decimal point) introduce features that are not backward compatible with earlier ODS versions, but InterBase 6 supports both ODS version 9 and ODS version 10. While it supports version 9, InterBase 6 uses ODS version 10. New features in this ODS that are not recognized by earlier software include 64-bit numerics and new datatypes: SQLDATE and SQLTIME. The old DATE datatype has been changed to TIMESTAMP—a more descriptive name.

When you create a new database or restore a backup file in the current version of InterBase, the resulting database file has the current ODS version.

IMPORTANT To upgrade the ODS of an older database, you must back it up using the backup utility for the version of the existing database and then restore it using the current version of InterBase.

Read-write and read-only databases

InterBase databases have two modes: read-only and read-write. At creation, all databases are both readable and writable: they are in *read-write mode*.

Read-write databases

To function in read-write mode, databases must exist on writable media and the **ibserver** process must have write access to the database file. For databases that are in read-write mode, this is true even when they are used only for reading because the transaction states are kept in an internal inventory data structure within the **.gdb** file. Therefore any transaction against the database requires the ability to write to the transaction inventory.

Under both Windows NT and UNIX, read-write database files must be writable by the user ID for the **ibserver** process. However, the operating environment or filesystem can be configured to create files that have limited file privileges by default. If you attempt to attach to a database and get an error of “unavailable database,” first check to see if the **.gdb** file’s permissions are such that the user ID of the **ibserver** process does not have write privilege on the database file.

Read-only databases

You can change InterBase databases to *read-only mode*. This provides enhanced security for databases by protecting them from accidental or malicious updates and enables distribution on read-only media such as CDROMs. Databases are always in read-write mode at creation time. This feature is independent of dialect. Any InterBase 6 or later database can be set to read-only mode.

You can use **gbak**, **gfix**, or IBConsole to change a database to read-only mode. (See **“Making a database read-only”** below.)

Properties of read-only databases

- In read-only mode, databases can be placed on CD-ROMs or in read-only filesystems as well as on read-write filesystems.
- Attempted INSERT, UPDATE, and DELETE operations on a read-only database generate an error. See the "Error Codes and Messages" chapter of the *Language Reference*.
- No metadata changes are allowed in read-only databases.
- Generators in a read-only database do not increment and are allowed only to return the current value. For example, in a read-only database, the following statement succeeds:

  ```
  SELECT GEN_ID(generator_name, 0) FROM table_name;
  ```

  The following statement fails with the error "attempted update on read-only database."

  ```
  SELECT GEN_ID(generator_name, 1) FROM table_name;
  ```

- External files accessed through a read-only database open in read-only mode, regardless of the file's permissions at the file system level.
- The read-only feature requires that both client and database be dialect 3, which is available only for InterBase 6 and later databases. This means that InterBase servers older than Version 6 cannot access read-only databases.

Making a database read-only

To change the mode of a database between read-write and read-only, you must be either its owner or SYSDBA and you must have exclusive access to a database.

You must have exclusive access to a database to change it to read-only mode.

From within InterBase, you can change a read-write database to read-only mode in any of three ways:

- In IBConsole, select the database, display its properties, and edit the mode. For more information, refer to **"Setting database properties" on page 118**.
- Use **gbak** to back up the database and restore it in read-only mode:

  ```
  gbak -create -mode read_only foo.gbk foo.gdb
  ```

- Use **gfix** to change the mode to read-only:

  ```
  gfix -mode read_only foo.gdb
  ```

IMPORTANT To set a database to read-only mode from any application that uses BDE, ODBC, or JDBC, use the *isc_action_svc_properties*() function in the InterBase Services API.

Tip To distribute a read-write database on a CD-ROM, back it up and put the *database*.**gbk** file on the CD-ROM. As part of the installation, restore the database to the user's hard disk.

Read-only with older InterBase versions

- A pre-6 InterBase client can access a read-only database to perform SELECTs. No other operation succeeds.
- If a version 6 client tries to set a pre-6 database to read-only mode, the server silently ignores the request. There is no way to make older databases read-only. You must upgrade them to version 6.

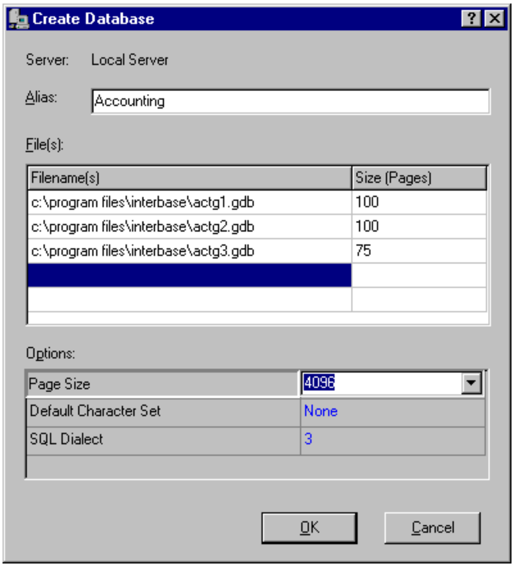
Creating databases

You can create databases on local and remote servers using IBConsole with the Create Database dialog.

You can use any of the following methods to access the Create Database dialog:

- In the Tree pane, select a server or anywhere in the branch under the desired server and choose **Database | Create Database**.
- In the Tree pane, right click the Databases branch under the desired server, and select Create Database from the context menu.

FIGURE 6.1 Create Database dialog



Server: Local Server

Alias: Accounting

File(s):

Filename(s)	Size (Pages)
c:\program files\interbase\actg1.gdb	100
c:\program files\interbase\actg2.gdb	100
c:\program files\interbase\actg3.gdb	75

Options:

Page Size	4096
Default Character Set	None
SQL Dialect	3

To create a database:

1. Ensure that the server indicated is correct. If it is not, cancel this dialog and re-initiate it under the correct server.
2. Type an Alias name for the new database in the Alias text field.
3. Enter one or more filenames which will make up the database, specifying the number of pages required for each file. To insert a new row into the Files table, move to the last row and column of the table and type Ctrl-Tab.

   When entering a filename, make sure to include the file path unless you wish to default the file to the working directory.

   Note Database files must reside on a local drive.
4. You can specify create options by entering a valid value, by clicking the option value and choosing a new value from a drop down list of values or by double-clicking the option value to rotate its value to the next in the list of values. For more information, see **"Database options"** below.

   To create a basic database without any options, leave all options blank.
5. Click OK to create the specified database.

IMPORTANT The alias name that you specify when creating a database references the necessary database file information associated with the database. When performing database configuration and maintenance, you need only specify the alias name, not the actual database filename. If the database spans multiple files, the server uses the header page of each file to locate additional files.

Database options

The database options that you can set are Page Size, Default Character Set, and SQL dialect.

Page size

InterBase supports database page sizes of 1024, 2048, 4096, and 8192 bytes. The default is 4096 bytes.

Default character set

See **"Character Set"** in Table **9.2** for a detailed explanation of character sets.

For more information about creating databases, see the *Language Reference*. See the *Data Definition Guide* for an explanation of character sets.

SQL dialect

An InterBase database SQL dialect determines how double quotes, large exact numerics, and certain datatypes such as SQL DATE, TIME, and TIMESTAMP are interpreted. In most cases you should choose dialect 3 in order to have access to all InterBase 6 features. For more information about dialects, refer to "Understanding SQL dialects" in *Getting Started*.

To change the database dialect, select the database in the Tree pane and choose Properties to display the Database Properties dialog. Click on the General tab and change the SQL dialect in the Options field.

Tip To suppress the display of system tables in IBConsole, deselect System Data from the View menu.

Dropping databases

You can drop databases using IBConsole. Dropping a database deletes the current database and database alias, removing both data and metadata.

To drop a database:

1. Select the database you wish to drop in the Tree pane.
2. Choose **Database | Drop Database** or select Drop Database from the Work pane.
3. A dialog asks you to confirm that you wish to delete the database. Click OK if you want to drop the selected database, otherwise click Cancel.

   Note A database can be dropped only by its creator or SYSDBA. A dropped database is removed from the list of databases maintained in the **interbas.ini** file.

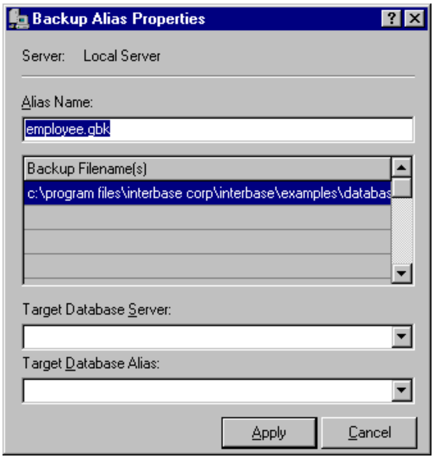
IMPORTANT Dropping a database deletes all data and metadata in the database.

Backup file properties

You can view and modify backup file information in IBConsole with the Backup Alias Properties dialog. You can access this dialog with either of the following methods:

- Expand Backup in the Tree pane, select a backup alias, and select Modify Backup Alias from the Work pane.
- Right-click a backup alias in the Tree pane and choose Modify Backup Alias from the context menu.

FIGURE 6.2 Backup alias properties



Server: Local Server

Alias Name: employee.gbk

Backup Filename(s)
c:\program files\interbase corp\interbase\examples\databas

Target Database Server:

Target Database Alias:

To edit backup file properties:

1. Enter a new backup alias name in the Alias Name text field.
2. Add, remove, or modify the backup filenames and corresponding file sizes associated with the backup in the backup files table. When specifying filenames, be sure to include the file path where the file is located.

   To add a new row to the backup files table, move to the last row and column of the table and type Ctrl-Tab. To remove a file from the backup file list, delete (blank out) the values in the table.
3. Select a server from the Target Database Server drop down list. You can also type the server name in the edit portion of the drop down list.
4. Select a database alias from the Target Database Alias drop down list. You can also type the alias name in the edit portion of the drop down list
5. Click Apply to save your changes.

Removing database backup files

You can remove database backup files in IBConsole with either of the following methods:

- Expand Backup in the Tree pane and select a backup alias and select Delete Alias from the Work pane.
- Right-click a backup alias in the Tree pane and choose Delete Alias from the context menu.

A dialog asks you to confirm that you wish to remove the selected backup file. Click OK if you want to delete the backup file, otherwise click Cancel.

Shadowing

InterBase lets you recover a database in case of disk failure, network failure, or accidental deletion of the database. The recovery method is called *disk shadowing*, or sometimes just *shadowing*. This chapter describes how to set up and use shadowing.This section describes the various tasks involved in shadowing, as well as the advantages and limitations of shadowing.

Tasks for shadowing

The main tasks in setting up and maintaining shadowing are as follows:

- Creating a shadow.

  Shadowing begins with the creation of a shadow. A shadow is an identical, physical copy of a database. When a shadow is defined for a database, changes to the database are written simultaneously to its shadow. In this way, the shadow always reflects the current state of the database. For information about the different ways to define a shadow, see **"Creating a shadow" on page 113**.

- Activating a shadow.

  If something happens to make a database unavailable, the shadow can be activated. *Activating* a shadow means it takes over for the database; the shadow becomes accessible to users as the main database. Activating a shadow happens either automatically or through the intervention of a database administrator, depending on how the shadow was defined. For more information about activating a shadow, see **"Activating a shadow" on page 117**.

- Deleting a shadow.

  If shadowing is no longer desired, it can be stopped by deleting the shadow. For more information about deleting a shadow, see **"Dropping a shadow" on page 117**.

- Adding files to a shadow.

  A shadow can consist of more than one file. As shadows grow in size, files can be added to accommodate the increased space requirements. For more information about adding shadow files, see **"Adding a shadow file" on page 118**.

Advantages of shadowing

Shadowing offers several advantages:

- Recovery is quick. Activating a shadow makes it available immediately.
- Creating a shadow does not require exclusive access to the database.
- Shadow files use the same amount of disk space as the database. Log files, on the other hand, can grow well beyond the size of the database.
- You can control the allocation of disk space. A shadow can span multiple files on multiple disks.
- Shadowing does not use a separate process. The database process handles writing to the shadow.
- Shadowing can run behind the scenes and needs little or no maintenance.

Limitations of shadowing

Shadowing has the following limitations:

- Shadowing is not an implementation of *replication*. Shadowing is one-way writing, duplicating every write operation on the master database. Client applications cannot access the shadow file directly. For information on replication, see **Chapter 11: "Data Replication"**
- Shadowing is useful only for recovery from hardware failures or accidental deletion of the database. User errors or software failures that corrupt the database are duplicated in the shadow.
- Recovery to a specific point in time is not possible. When a shadow is activated, it takes over as a duplicate of the database. Shadowing is an "all or nothing" recovery method.
- Shadowing can occur only to a local disk. Shadowing to a NFS filesystem or mapped drive is not supported. Shadowing to tape or other media is unsupported.

Creating a shadow

A shadow is created with the CREATE SHADOW statement in SQL. Because this does not require exclusive access, it can be done without affecting users. For detailed information about CREATE SHADOW, see the *Language Reference*.

Before creating a shadow, consider the following topics:

- The location of the shadow

  A shadow should be created on a different disk from where the main database resides. Because shadowing is intended as a recovery mechanism in case of disk failure, maintaining a database and its shadow on the same disk defeats the purpose of shadowing.

- Distributing the shadow

  A shadow can be created as a single disk file called a shadow file or as multiple files called a shadow set. To improve space allocation and disk I/O, each file in a shadow set can be placed on a different disk.

- User access to the database

  If a shadow becomes unavailable, InterBase can either deny user access to the database until shadowing is resumed, or allow access even though database changes are not being shadowed. Depending on which database behavior is desired, the database administrator creates a shadow either in auto mode or in manual mode. For more information about these modes, see **"Auto mode and manual mode" on page 115**.

- Automatic shadow creation

  To ensure that a new shadow is automatically created, create a conditional shadow. For more information, see **"Conditional shadows,"** in this chapter.

  The next sections describe how to create shadows with various options:

- Single-file or multifile shadows
- Auto or manual shadows
- Conditional shadows

  These choices are not mutually exclusive. For example, you can create a single-file, conditional shadow in manual mode.

Creating a single-file shadow

To create a single-file shadow for database **employee.gdb**, enter:

```
SQL> CREATE SHADOW 1 '/usr/interbase/examples/employee.shd';
```

The name of the shadow file is **employee.shd**, and it is identified by the number 1. Verify that the shadow has been created by using the **isql** command SHOW DATABASE:

```
SQL> SHOW DATABASE;
    Database: employee.gdb
     Shadow 1: '/usr/interbase/examples/employee.shd' auto
    PAGE_SIZE 4096
    Number of DB pages allocated = 392
    Sweep interval = 20000
```

The page size of the shadow is the same as that of the database.

Creating a multifile shadow

If your database is large, you can shadow it to a multifile shadow, spreading the shadow files over several disks. To create a multifile shadow, specify the name and size of each file in the shadow set. As with multifile databases, you have the option of specifying the size of secondary files in either of two ways: specify the page on which each secondary file starts, or specify the length in database pages of each file. When you specify the size using the LENGTH keyword, do not specify the length of the final file. InterBase sizes the final file dynamically, as needed.

For example, the following example creates a shadow set consisting of three files. The primary file, **employee.shd**, is 10,000 database pages in length. The second file is 20,000 database pages long, and the final file grows as needed.

```
SQL> CREATE SHADOW 1 'employee.shd' LENGTH 10000
```

```
CON> FILE 'emp2.shd' LENGTH 20000
CON> FILE 'emp3.shd';
```

Instead of specifying the page length of secondary files, you can specify their starting page. The following example creates the same shadows as the previous example:

```
SQL> CREATE SHADOW 1 'employee.shd'
CON> FILE 'emp1.shd' STARTING AT 10000
CON> FILE 'emp2.shd' STARTING AT 30000;
```

In either case, you can use SHOW DATABASE to verify the file names, page lengths, and starting pages for the shadow just created:

```
SQL> SHOW DATABASE;
   Database: employee.gdb
    Shadow 1: '/usr/interbase/examples/employee.shd' auto length 10000
    file /usr/interbase/examples/emp1.shd length 2000 starting 10000
    file /usr/interbase/examples/emp2.shd length 2000 starting 30000
   PAGE_SIZE 4096
   Number of DB pages allocated = 392
   Sweep interval = 20000
```

Note The page length you allocate for secondary shadow files need not correspond to the page length of the database's secondary files. As the database grows and its first shadow file becomes full, updates to the database automatically overflow into the next shadow file.

Auto mode and manual mode

A shadow can become unavailable for the same reasons a database becomes unavailable (disk failure, network failure, or accidental deletion). If a shadow becomes unavailable, and it was created in *auto mode*, database operations continue automatically without shadowing. If a shadow becomes unavailable, and it was created in *manual mode*, further access to the database is denied until the database administrator intervenes. The benefits of auto mode and manual mode are compared in the following table:

Mode	Advantage	Disadvantage
Auto	Database operation is uninterrupted	Creates a temporary period when the database is not shadowed The database administrator might be unaware that the database is operating without a shadow
Manual	Prevents the database from running unintentionally without a shadow	Database operation is halted until the problem is fixed Needs intervention of the database administrator

TABLE 6.1 Auto vs. manual shadows

AUTO MODE

The AUTO keyword directs the CREATE SHADOW statement to create a shadow in auto mode:

```
SQL> CREATE SHADOW 1 AUTO 'employee.shd';
```

Auto mode is the default, so omitting the AUTO keyword achieves the same result.

In AUTO mode, database operation is uninterrupted even though there is no shadow. To resume shadowing, it might be necessary to create a new shadow. If the original shadow was created as a conditional shadow, a new shadow is automatically created. For more information about conditional shadows, see **"Conditional shadows" on page 117**.

MANUAL MODE

The MANUAL keyword directs the CREATE SHADOW statement to create a shadow in manual mode:

```
SQL> CREATE SHADOW 1 MANUAL 'employee.shd';
```

Manual mode is useful when continuous shadowing is more important than continuous operation of the database. When a manual-mode shadow becomes unavailable, further attachments to the database are prevented. To allow database attachments again, the database owner or SYSDBA must enter the following command:

```
gfix -kill database
```

This command deletes metadata references to the unavailable shadow corresponding to *database*. After deleting the references, a new shadow can be created if shadowing needs to resume.

Conditional shadows

You can define a shadow such that if it replaces a database, the server creates a new shadow file, allowing shadowing to continue uninterrupted. A shadow defined with this behavior is called a *conditional shadow.*

To create a conditional shadow, specify the CONDITIONAL keyword with the CREATE SHADOW statement. For example,

```
SQL> CREATE SHADOW 3 CONDITIONAL 'atlas.shd';
```

Creating a conditional file directs InterBase to automatically create a new shadow. This happens in either of two cases:

- The database or one of its shadow files becomes unavailable.
- The shadow takes over for the database due to hardware failure.

Activating a shadow

When a database becomes unavailable, database operations are resumed by activating the shadow. To do so, log in as SYSDBA or the database owner and use **gfix** with the **–activate** (or **–a**) option.

IMPORTANT Before activating a shadow, check that the main database is unavailable. If a shadow is activated while the main database is available, the shadow can be corrupted by existing attachments to the main database.

To activate a shadow, specify the path name of its primary file. For example, if database **employee.gdb** has a shadow named **employee.shd**, enter:

```
gfix –a employee.shd
```

After a shadow is activated, you should change its name to the name of your original database. Then, create a new shadow if shadowing needs to continue and if another disk drive is available.

Dropping a shadow

To stop shadowing, use the shadow number as an argument to the DROP SHADOW statement. For example,

```
SQL> DROP SHADOW 1
```

If you need to look up the shadow number, use the **isql** command SHOW DATABASE.

IMPORTANT DROP SHADOW deletes shadow references from a database's metadata, as well as the physical files on disk. Once the files have been removed from disk, there is no opportunity to recover them. However, a shadow is merely a copy of an existing database, so the new shadow is identical to the dropped shadow.

Adding a shadow file

If a database is expected to increase in size, consider adding files to its shadow. To add a shadow file, first use DROP SHADOW to delete the existing shadow, then use CREATE SHADOW to create a multifile shadow.

The page length you allocate for secondary shadow files need not correspond to the page length of the database's secondary files. As the database grows and its first shadow file becomes full, updates to the database automatically overflow into the next shadow file.

Setting database properties

The Database Properties dialog enables you to display and configure certain database settings. You can access the Database Properties dialog by any of the following methods:

- Select a connected database (or any branch under the database hierarchy) in the Tree pane and choose **Database | Properties**.
- Select a connected database in the Tree pane and click Properties in the Work pane.
- Right-click a connected database in the Tree pane and choose Properties from the context menu.

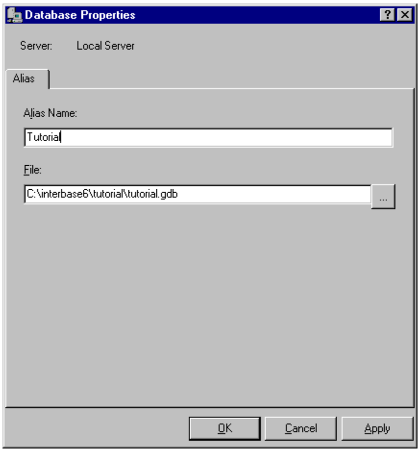
Note If you want to change the database file name, the database must be disconnected before you access the Database Properties dialog. The General tab of this dialog is not available for disconnected databases.

The Database Properties dialog contains two tabs, Alias and General.

Alias tab

The Alias tab of the Database Properties dialog is where you can specify an alias name for a database as well as the file path and file name of the selected database.

FIGURE 6.3 Database Properties - Alias tab



Server: Local Server

Alias Name: Tutorial

File: C:\interbase6\tutorial\tutorial.gdb

To edit database alias settings:

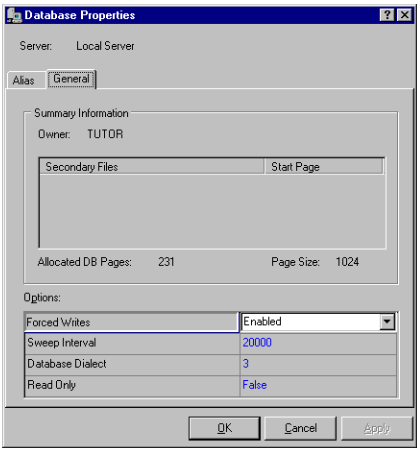
1. Enter the alias name of the database in the Alias Name text field.
2. Enter database file name, including the path where the file is located, in the File text field. If you prefer, you can also click the browse button ... to locate the file you want.
3. If you need to view or configure the general database settings, click the General tab and see **"General tab"** below for further information.
4. Once you are finished making changes to the database properties click Apply to save your changes, otherwise click Cancel.

General tab

The General tab of the Database Properties dialog is where you can view such database settings as the database owner, secondary files and their start pages, the number of allocated database pages and the page size. You can also set such options as Forced Writes, Sweep Interval, SQL Dialect and Read Only.

Note If you want to change the database file name, the database must be disconnected before you access the Database Properties dialog. The General tab of this dialog is not available for disconnected databases.

FIGURE 6.4 Database Properties - General tab



Database Properties

Server: Local Server

Summary Information

Owner: TUTOR

Secondary Files	Start Page

Allocated DB Pages: 231

Page Size: 1024

Options:

Forced Writes	Enabled
Sweep Interval	20000
Database Dialect	3
Read Only	False

To edit database general options:

1. Choose option values in the Options table. You can specify options by clicking the option value and entering a new value, by choosing a new value from a drop down list of values or by double-clicking the option value to rotate its value to the next in the list of values.
2. If you need to view or configure the database alias settings, click the Alias tab and see **"Alias tab"** above for further information.
3. Once you are finished making changes to the database properties click Apply to save your changes, otherwise click Cancel.

Option	Value
Forced Writes	Option values are Enabled and Disabled. See **"Forced writes vs. buffered writes" on page 126** for further information on forced writes.
Sweep Interval	The sweep interval is the number of transactions that will occur before an automatic database sweep takes place. You can enter any positive number for the sweep interval, or zero to disable the automatic sweep. See **"Sweep interval and automated housekeeping" on page 121** for further information on setting the sweep interval.
Database dialect	An InterBase database SQL dialect determines how double quotes, large exact numerics, and certain datatypes such as SQL DATE, TIME, and TIMESTAMP are interpreted. In most cases you should choose dialect 3 in order to have access to all InterBase 6 features.
Read Only	Option values are True and False. To make the database read only set the Read Only option to True. This prevents users from performing any DML or updates to the database. The default setting for this option is False. See **"Making a database read-only" on page 106** for more information.

TABLE 6.2 General options

Sweep interval and automated housekeeping

Sweeping a database is a systematic way of removing outdated records. Periodic sweeping prevents a database from growing too large. However, sweeping can also slow system performance.

As a database administrator, you can tune database sweeping, balancing its advantages and disadvantages to best satisfy users' needs.

Overview of sweeping

InterBase uses a multigenerational architecture. This means that multiple versions of data records are stored directly on the data pages. When a record is updated or deleted, InterBase keeps a copy of the old state of the record and creates a new version. This can increase the size of a database.

GARBAGE COLLECTION

To limit the growth of the database, InterBase performs garbage collection by sweeping the database. This process frees up space allocated to outdated record versions. Whenever a transaction accesses a record, outdated versions of that record are collected. Records that were rolled back are not collected. To guarantee that all outdated records are collected, including those that were rolled back, InterBase periodically sweeps the database.

AUTOMATIC HOUSEKEEPING

If a transaction is left in an active (unresolved) state, this is an "interesting" transaction. In a given database's transaction inventory, the first transaction with a state other than committed is known as the Oldest Interesting Transaction (OIT). Automatic housekeeping occurs when the difference between the OIT and the oldest active transaction (OAT) is greater than the sweep interval. By default, this sweep interval is 20,000, but it is configurable (see **"Setting the sweep interval" on page 123**).

Note It is a subtle but important distinction that the automatic sweep does *not* necessarily occur every 20,000 transactions. It is only when the *difference* between the OIT and OAT reaches the threshold. If every transaction to the database is committed promptly, then this difference it is not likely to be great enough to trigger the automatic sweep.

The InterBase server process initiates a special thread to perform this sweep asynchronously, so that the client process can continue functioning, unaffected by the amount of work done by the sweep.

Tip Sweeping a database is not the only way to perform systematic garbage collection. Backing up a database achieves the same result, because the InterBase server must read every record, an action that forces garbage collection throughout the database. As a result, regularly backing up a database can reduce the need to sweep. This enables you to maintain better application performance. For more information about the advantages of backing up and restoring, see **"Benefits of backup and restore" on page 143**.

CONFIGURING SWEEPING

You are able to control several aspects of database sweeping. You can:

- Change the automatic sweep interval.
- Disable automatic sweeping.
- Sweep a database immediately.

The first two functions are performed in the Database Properties dialog. The last is performed with a sweep menu command and is explained in **"Performing an immediate database sweep" on page 123**.

Setting the sweep interval

To set the automatic sweep threshold to *n* transactions:

```
gfix -h n
```

Sweeping a database can affect transaction start-up if rolled back transactions exist in the database. As the time since the last sweep increases, the time for transaction start-up can also increase. Lowering the sweep interval can help reduce the time for transaction start-up.

On the other hand, frequent database sweeps can reduce application performance. Raising the sweep interval could help improve overall performance. The database administrator should weigh the issues for the affected applications and decide whether the sweep interval provides the desired database performance.

To set the sweep interval with IBConsole, refer to **"Setting database properties" on page 118**.

Tip Unless the database contains many rolled back transactions, changing the sweep interval has little effect on database size. As a result, it is more common for a database administrator to tune the database by disabling sweeping and performing it at specific times. These activities are described in the next two sections.

Disabling automatic sweeping

To disable automatic sweeping, set the sweep threshold to zero (0). Disabling automatic sweeping is useful if:

- Maximum throughput is important. Transactions are never delayed by sweeping.
- You want to schedule sweeping at specific times. You can manually sweep the database at any time. It is common to schedule sweeps at a time of least activity on the database server, to avoid competing for resources with clients.

To disable automatic sweeping with IBConsole, refer to **"Setting database properties" on page 118**.

Performing an immediate database sweep

You can perform an immediate database sweep with any of the following methods:

- Right click a connected database in the Tree pane and choose **Maintenance | Sweep** from the context menu
- Select a connected database in the Tree pane and click Sweep in the Work pane

- enter the following command: `gfix -s`

  This operation runs an immediate sweep of the database, releasing space held by records that were rolled back and by out-of-date record versions. Sweeps are also done automatically at a specified interval.

  Sweeping a database does not strictly require it to be shut down. You can perform sweeping at any time, but it can impact system performance and should be done when it inconveniences users the least.

  If a sweep is performed as an exclusive operation on the database, there is additional tuning that the procedure performs. As long as there are no outstanding active transactions, the sweep updates the state of data records and the state of the inventory of past transactions. Non-committed transactions are finally rendered obsolete, and internal data structures need not track them in order to maintain snapshots of database versions. The benefit of this is a reduction of memory use, and a noticeable performance improvement.

Configuring the database cache

The *database cache* consists of all database pages (also called buffers) held in memory at one time. *Database cache size* is the number of database pages. You can set the default size of the database cache at three levels:

- server level: applies to all databases
- database level: applies only to a single database (using **gfix** to set the size for a specific database)
- connection level: applies only to a specific **isql** connection

We recommend setting cache size at the database level rather than at the server level. This reduces the likelihood of inappropriate database cache sizes.

In a SuperServer installation (Windows, Linux, or UNIX), every database on a server requires RAM equal to the cache size (number of database pages) times the page size. By default, the cache size is 2048 pages per database and the page size is 4KB. Thus, a single database running at the default setting requires 8MB of memory, but three such databases require 24MB of memory.

In Classic installations, the amount of memory required by a database depends on the number of client attachments. Each client is allotted 75 cache pages, so memory usage is calculated by:

```
cache size × number of attachments × 75
```

Default cache size per database

The **buffers** parameter of the **gfix** utility sets the default number of cache pages for a specific database:

```
gfix -buffers n database_name
```

This sets the number of cache pages for the specified database to *n*, overriding the server value, which by default is 2048 pages.

To run **gfix**, you must be either SYSDBA or the owner of the database.

Default cache size per ISQL connection

To configure the number of cache pages for the duration of one **isql** connection, invoke **isql** with the following option:

```
isql -c n database_name
```

n is the number of cache pages to be used as the default for the session; *n* overrides any values set by DATABASE_CACHE_PAGES or **gfix** and must be greater than 9.

A CONNECT statement entered in an **isql** query accepts the argument CACHE n. (Refer to the discussion of CONNECT in the *Language Reference* manual for a full description of the CONNECT function). For example:

```
ISQL> CONNECT database_name CACHE n;
```

The value *n* can be any positive integer number of database pages. If a database cache already exists in the server because of another attachment to the database, the cache size is increased only if *n* is greater than current cache size.

Setting cache size in applications

InterBase API: use the *isc_dpb_num_buffers* parameter to set cache size in a database parameter buffer (DPB).

IBX: use the *num_buffers* parameter to set cache size in the TIBDatabase's parameter list. For example: *num_buffers*=250. For the parameter to be parsed correctly, there must be no spaces around the = sign.

Default cache size per server

For SuperServer installations, you can configure the default number of pages used for the database caches. By default, the database cache size is 2048 pages per database. You can modify this default by changing the value of DATABASE_CACHE_PAGES in the **isc_config** (UNIX) or **ibconfig** (Windows) file. When you change this setting, it applies to every active database on the server.

You can also set the default cache size for each database using the **gfix** utility. This approach permits greater flexibility, and reduces the risk that memory is overused, or that database caches are too small. *We strongly recommend that you use* **gfix** *to set cache size rather than* DATABASE_CACHE_PAGES.

Verifying cache size

To verify the size of the database cache currently in use, execute the following commands in **isql**:

```
ISQL> CONNECT database_name;
ISQL> SET STATS ON;
ISQL> COMMIT;
Current memory = 415768
Delta memory = -2048
Max memory = 419840
Elapsed time = 0.03 sec
Buffers = 2048
Reads = 0
Writes 2
Fetches = 2
ISQL> QUIT;
```

The empty COMMIT command prompts **isql** to display information about memory and buffer usage. The “Buffers” line specifies the size of the cache for that database.

Forced writes vs. buffered writes

When an InterBase Server performs forced writes (also referred to as synchronous writes), it physically writes data to disk whenever the database performs an (internal) write operation.

If forced writes are not enabled, then even though InterBase performs a write, the data may not be physically written to disk, since operating systems buffer disk writes. If there is a system failure before the data is written to disk, then information can be lost.

Performing forced writes ensures data integrity and safety, but slow performance. In particular, operations that involve data modification are slower.

Forced writes are enabled or disabled in the Database Properties dialog. For more information, refer to **"Setting database properties" on page 118**.

Validation and repair

In day-to-day operation, a database is sometimes subjected to events that pose minor problems to database structures. These events include:

- Abnormal termination of a database application. This does not affect the integrity of the database. When an application is canceled, committed data is preserved, and uncommitted changes are rolled back. If InterBase has already assigned a data page for the uncommitted changes, the page might be considered an orphan page. Orphan pages are unassigned disk space that should be returned to free space.
- Write errors in the operating system or hardware. These usually create a problem with database integrity. Write errors can cause data structures such as database pages and indexes to become broken or lost. These corrupt data structures can make committed data unrecoverable.

Validating a database

You should validate a database:

- Whenever a database backup is unsuccessful.
- Whenever an application receives a "corrupt database" error.
- Periodically, to monitor for corrupt data structures or misallocated space.
- Any time you suspect data corruption.

Database validation requires exclusive access to the database. Shut down a database to acquire exclusive access. If you do not have exclusive access to the database, you get the error message:

```
OBJECT database_name IS IN USE
```

To shut down a database, refer to the directions in **"Shutting down a database" on page 131**.

Validating a database using gfix

To validate a database using gfix, follow these steps:

1. Enter the following command:

```
gfix -v
```

2. If you suspect you have a corrupt database, make a copy of your database using an OS command (**gbak** will not back up corrupt data).
3. Use the **gfix** command to mark corrupt structures in the copied database:

```
gfix -m
```

4. If gfix reports any checksum errors, validate and repair the database again, ignoring any checksum errors:

```
gfix -v -i
```

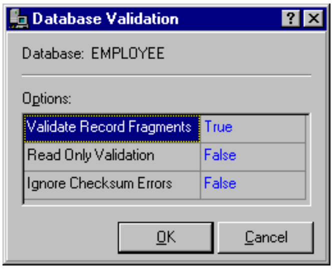
It may be necessary to validate a database multiple times to correct all the errors.

Validating a database using IBConsole

To validate a database using IBConsole, access the Database Validation dialog by any of the following methods:

- Select a disconnected database in the Tree pane and choose Validation in the Work pane.
- Right-click a disconnected database in the Tree pane and choose Validation from the context menu.
- Select **Database | Maintenance | Validation**.

FIGURE 6.5 Database Validation dialog



Database: EMPLOYEE

Options:

Validate Record Fragments	True
Read Only Validation	False
Ignore Checksum Errors	False

To validate database:

1. Check that the database indicated is correct. If it is not, cancel this dialog and re-initiate the Database Validation dialog under the correct database.
2. Specify validation options by entering a valid value, by clicking the option value and choosing a new value from a drop down list of values or by double-clicking the option value to rotate its value to the next in the list of values.
3. Click OK if you want to proceed with the validation, otherwise click Cancel.

When IBConsole validates a database it verifies the integrity of data structures. Specifically, it does the following:

- Reports corrupt data structures
- Reports misallocated data pages
- Returns orphan pages to free space

Option	Value
Validate Record Fragments	Option values are True and False. By default, database validation reports and releases only page structures. If the Validate Record Fragments option is set to True, validation reports and releases record structures as well as page structures.
Read Only Validation	Option values are True and False. By default, validating a database updates it, if necessary. To prevent updating, set the Read Only Validation option to True.
Ignore Checksum Errors	Option values are True and False. A checksum is a page-by-page analysis of data to verify its integrity. A bad checksum means that a database page has been randomly overwritten (for example, due to a system crash). Checksum errors indicate data corruption. To repair a database that reports checksum errors, set the Ignore Checksum Errors option to True. This enables IBConsole to ignore checksums when validating a database. Ignoring checksums allows successful validation of a corrupt database, but the affected data may be lost.

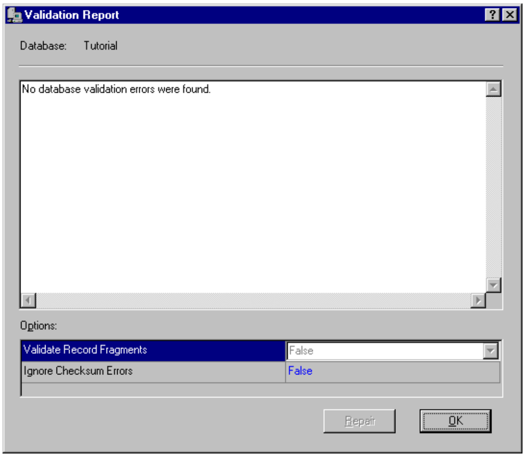
TABLE 6.3 Validation options

IMPORTANT Even if you can restore a mended database that reported checksum errors, the extent of data loss may be difficult to determine. If this is a concern, you may want to locate an earlier backup copy and restore the database from it.

Repairing a corrupt database

If a database contains errors, they are displayed in the following dialog:

FIGURE 6.6 Validation report dialog



Database: Tutorial

No database validation errors were found.

Options:

Validate Record Fragments	False
Ignore Checksum Errors	False

The errors encountered are summarized in the text display area. The repair options you selected in the Database Validation dialog are selected in this dialog also.

To repair the database, choose Repair. This fixes problems that cause records to be corrupt and mark corrupt structures. In subsequent operations (such as backing up), InterBase ignores the marked records.

Note Some corruptions are too serious for IBConsole to correct. These include corruptions to certain strategic structures, such as space allocation pages. In addition, IBConsole cannot fix certain checksum errors that are random by nature and not specifically associated with InterBase.

If you suspect you have a corrupt database, perform the following steps:

1. Make a copy of the database using an operating-system command. Do not use the IBConsole Backup utility or the **gbak** command, because they cannot back up a database containing corrupt data. If IBConsole reports any checksum errors, validate and repair the database again, setting the Ignore Checksum Error option to True.
2. It may be necessary to validate a database multiple times to correct all the errors. Validate the database again, with the Read Only Validation option set to True.

Note Free pages are no longer reported, and broken records are marked as damaged. Any records marked during repair are ignored when the database is backed up.

3. Back up the mended database with IBConsole or **gbak**. At this point, any damaged records are lost, since they were not included during the back up. For more information about database backup, see **Chapter 7, "Database Backup and Restore."**
4. Restore the database to rebuild indexes and other database structures. The restored database should now be free of corruption.

Verify that restoring the database fixed the problem by validating the restored database with the Read Only Validation option set to True.

Shutting down and restarting databases

Maintaining a database often involves shutting it down. Only the SYSDBA or the owner of a database (the user who created it) can shut it down. The user who shuts down the database then has exclusive access to the database.

Exclusive access to a database is required to:

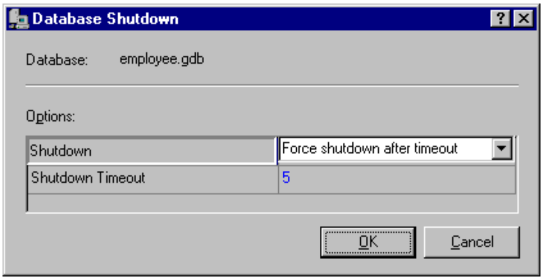
- Validate and repair the database.
- Add or drop a foreign key on a table in the database.
- Add a secondary database file.

After a database is shut down, the database owner and SYSDBA are still able to connect to it, but any other user attempting to connect gets an error message stating that the database is shut down.

Shutting down a database

To shut down a database, select a connected database from the Tree pane and choose Shutdown in the Work pane or choose **Database | Maintenance | Shutdown**. The Database Shutdown dialog appears:

FIGURE 6.7 Database shutdown dialog



Database: employee.gdb

Options:

Shutdown	Force shutdown after timeout
Shutdown Timeout	5

Shutdown timeout options

You can specify a timeout value by selecting a new value from the drop down list of values or by typing the value in the edit portion of the drop down list. Timeout values can range from 1 minute to 500 minutes.

Shutdown options

You can specify shutdown options by selecting a new value from the drop down list of values. Shutdown option values include: Deny New Connections While Waiting, Deny New Transactions While Waiting, and Force Shutdown After Timeout.

DENY NEW CONNECTIONS WHILE WAITING

This option allows all existing database connections to complete their operations unaffected. IBConsole shuts down the database after all processes disconnect from the database. At the end of the timeout period, if there are still active connections, then the database is not shut down.

This prevents any new processes from connecting to the database during the timeout period. This enables current users to complete their work, while preventing others from beginning new work.

Suppose the SYSDBA needs to shut down database **orders.gdb** at the end of the day (five hours from now) to perform routine maintenance. The Marketing department is currently using the database to generate important sales reports.

In this case, the SYSDBA would shut down **orders.gdb** with the following parameters:

- Deny New Connections.
- Timeout of 300 minutes (five hours).

These parameters specify to deny any new database connections and to shut down the database any time during the next five hours when there are no more active connections.

Any users who are already connected to the database are able to finish processing their sales reports, but new connections are denied. During the timeout period, the SYSDBA sends out periodic broadcast messages asking users to finish their work by 6 p.m.

When all users have disconnected, the database is shut down. If all users have not disconnected after five hours, then the database is not shut down. Because the shutdown is not critical, it is not forced.

It would be inappropriate to deny new transactions, since generating a report could require several transactions, and a user might be disconnected from the database before completing all necessary transactions. It would also be inappropriate to force shutdown, since it might cause users to lose work.

DENY NEW TRANSACTIONS WHILE WAITING

This option allows existing transactions to run to normal completion. Once transaction processing is complete, IBConsole shuts down the database. Denying new transactions also denies new database connections. At the end of the timeout period, if there are still active transactions, then the database is not shut down.

This is the most restrictive shutdown option, since it prevents any new transactions from starting against the database. This option also prevents any new connections to the database.

Suppose the SYSDBA needs to perform critical operations that require shutdown of the database **orders.gdb**. This is a database used by dozens of customer service representatives throughout the day to enter new orders and query existing orders.

At 5 p.m., the SYSDBA initiates a database shutdown of **orders.gdb** with the following parameters:

- Deny New Transactions.
- Timeout of 60 minutes.

These parameters deny new transactions for the next hour. During that time, users can complete their current transactions before losing access to the database. Simply denying new connections would not be sufficient, since the shutdown cannot afford to wait for users to disconnect from the database.

During this hour, the SYSDBA sends out periodic broadcast messages warning users that shutdown is happening at 6 p.m and instructs them to complete their work. When all transactions have been completed, the database is shut down.

After an hour, if there are still any active transactions, IBConsole cancels the shutdown. Since the SYSDBA needs to perform database maintenance, and has sent out numerous warnings that a shutdown is about to occur, there is no choice but to force a shutdown.

FORCE SHUTDOWN AFTER TIMEOUT

With this option, there are no restrictions on database transactions or connections. As soon as there are no processes or connections to the database, IBConsole shuts down the database. At the end of the timeout period, if there are still active connections, IBConsole rolls back any uncommitted transactions, disconnects any users, and shuts down the database.

If critical database maintenance requires a database to be shut down while there are still active transactions, the SYSDBA can force shut down. This step should be taken only if broadcast messages have been sent out to users that shutdown is about to occur. If users have not heeded repeated warnings and remain active, then their work is rolled back.

This option does not deny new transactions or connections during the timeout period. If, at any time during the timeout period, there are no connections to the database, IBConsole shuts down the database.

IMPORTANT Forcing database shutdown interferes with normal database operations, and should only be used after users have been given appropriate broadcast notification well in advance.

Restarting a database

After a database is shut down, it must be restarted (brought back online) before users can access it.

To restart a database, select a previously shut down database from the Tree pane and choose **Database | Maintenance | Database Restart** or choose Database Restart in the Work pane. The currently selected database is brought back online immediately.

Limbo transactions

When committing a transaction that spans multiple databases, InterBase automatically performs a two-phase commit. A *two-phase commit* guarantees that the transaction updates either all of the databases involved or none of them—data is never partially updated.

Note The Borland Database Engine (BDE), as of version 4.5, does not exercise the two-phase commit or distributed transactions capabilities of InterBase, therefore applications using the BDE never create limbo transactions.

In the first phase of a two-phase commit, InterBase prepares each database for the commit by writing the changes from each *subtransaction* to the database. A subtransaction is the part of a multi-database transaction that involves only one database. In the second phase, InterBase marks each subtransaction as committed in the order that it was prepared.

If a two-phase commit fails during the second phase, some subtransactions are committed and others are not. A two-phase commit can fail if a network interruption or disk crash makes one or more databases unavailable. Failure of a two-phase commit causes limbo transactions, transactions that the server does not know whether to commit or roll back.

It is possible that some records in a database are inaccessible due to their association with a transaction that is in a limbo state. To correct this, you must recover the transaction using IBConsole. *Recovering* a limbo transaction means committing it or rolling it back. Use **gfix** to recover transactions.

Recovering transactions

You can recover transactions by any of the following methods:

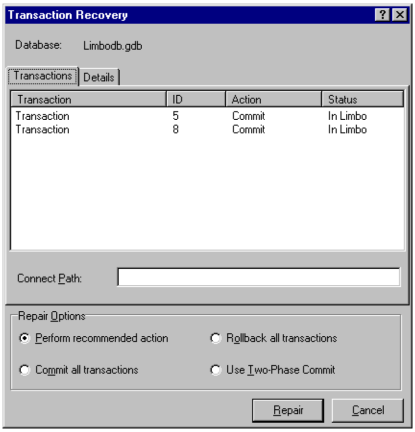
- Select a connected database in the Tree pane and choose Transaction Recovery in the Work pane or choose **Database | Maintenance | Transaction Recovery**.
- Right-click a connected database in the Tree pane and choose **Maintenance | Transaction Recovery** from the context menu.

The Transaction Recovery dialog contains two tabs, Transactions and Details. The Transactions tab displays a list of limbo transactions that can then be operated upon to recover - that is, to commit or roll back. You can also seek suggested recovery actions and set current actions to perform on the selected limbo transactions. The Details tab displays detailed information about a selected transaction.

Transaction tab

All the pending transactions in the database are listed in the text area of the Transactions tab. You can rollback, commit, or perform a two-phase commit on such transactions.

FIGURE 6.8 Transaction Recovery - limbo transactions



Database: Limbodb.gdb

Transaction	ID	Action	Status
Transaction	5	Commit	In Limbo
Transaction	8	Commit	In Limbo

Connect Path:

Repair Options

● Perform recommended action ○ Rollback all transactions

○ Commit all transactions ○ Use Two-Phase Commit

To recover limbo transactions:

1. Select a limbo transaction in the table.
2. The Connect Path text field displays the current path of the database file for the selected transaction, if it is a multi-database transaction. You can change the target database path, if necessary, by overwriting the current path.

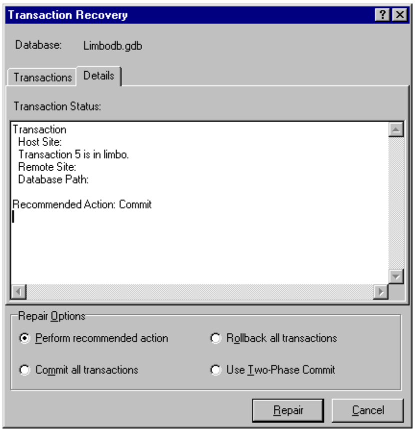
   The information on the path to the database was stored when the client application attempted the commit. It is possible that the path and network protocol from that machine does not work from the client which is now running IBConsole. Before attempting to roll back or commit any transaction, confirm the path of all involved databases is correct.

   When entering the current path, be sure to include the server name and separator indicating communication protocol. To use TCP/IP, separate the server and directory path with a colon (:). To use NetBEUI or SPX, precede the server name with either a double back slash (\\) or a double slash (//), and then separate the server name and directory path with either a back slash or a slash.
3. If you want to continue with the transaction recovery process select a repair option and click Repair, otherwise click Cancel. To determine the recommended action, click on the transaction and select the Details tab. For further information about transaction recovery suggestions, see **"Details tab"** below.

Details tab

The Details tab displays the host server, the remote server, database path, and recommended action: either commit or rollback. If you want to continue with the transaction recovery process select a repair option and click Repair, otherwise click Cancel.

FIGURE 6.9 Transaction Recovery - Details



Database: Limbodb.gdb

Transaction Status:

```
Transaction
 Host Site:
 Transaction 5 is in limbo.
 Remote Site:
 Database Path:

Recommended Action: Commit
```

Repair Options

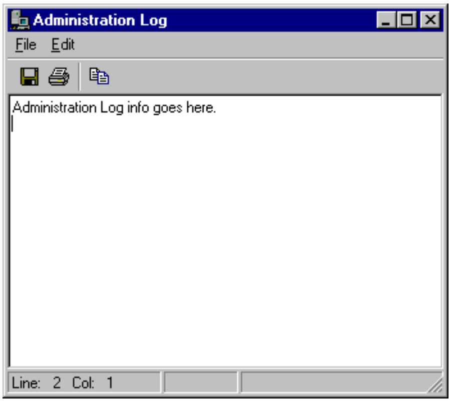
- ● Perform recommended action
- ○ Rollback all transactions
- ○ Commit all transactions
- ○ Use Two-Phase Commit

Viewing the administration log

IBConsole displays the administration log file in a standard text display window, the Administration Log dialog, which can be accessed by any of the following methods:

- Select a server (or any branch under the server hierarchy) in the Tree pane and choose **Server | Administration Log**.
- Select a server in the Tree pane and click View Administration Log in the Actions tab of the Work pane.
- Right-click the desired server in the Tree pane and choose Administration Log from the context menu.
- Select InterBase Server Aliases in the Tree pane to display a list of registered servers in the Summary tab of the Work pane. Right-click a server in the Work pane and choose Administration Log from the context menu.

FIGURE 6.10 Administration Log dialog



Administration Log info goes here.

The standard text display window enables you to search for specific text, save the text to a file, and print the text. For an explanation of how to use the standard text display window, see **"Standard text display window" on page 37.**

gfix command-line tool

The **gfix** tool performs a number of maintenance activities on a database, including the following:

- Database shutdown
- Changing database mode to read-only or read-write
- Changing the dialect of a database
- Setting cache size at the database level
- Committing limbo transactions
- Mending databases and making minor data repairs
- Sweeping databases
- Displaying, committing, or recovering limbo transactions

To run **gfix**, you must attach as either SYSDBA or the owner of the database. Most of these actions can also be performed through IBConsole.

Syntax

```
gfix [options] db_name
```

Options In the OPTION column of the following table, only the characters outside the brackets ([ ]) are required. You can specify additional characters up to and including the full option name. To help identify options that perform similar functions, the TASK column indicates the type of activity associated with an option.

Option	Task	Description
-ac[tivate]	Activate shadows	Activate shadows when the database dies. NOTE: syntax is gfix -ac (no database name)
–at[**tach**] *n*	Shutdown	Used with **–shut** to prevent new database connections during timeout period of *n* seconds; shutdown is canceled if there are still processes connected after *n* seconds
–b[**uffers**] *n*	Cache buffers	Sets default cache buffers for the database to *n* pages
–ca[**che**] *n*		Reserved for future implementation
–c[**ommit**] **{ID | all}**	Transaction recovery	Commits limbo transaction specified by **ID** or commit all limbo transactions
–force *n*	Shutdown	Used with **–shut** to force shutdown of a database after *n* seconds; this is a drastic solution that should be used with caution
–f[**ull**]	Data repair	Used with **–v** to check record and page structures, releasing unassigned record fragments
–h[**ousekeeping**] *n*	Sweeping	Changes automatic sweep threshold to *n* transactions • Setting *n* to 0 disables sweeping • Default threshold is 20,000 transactions (see **"Overview of sweeping" on page 121**) • Exclusive access not needed
–i[**gnore**]	Data repair	Ignores checksum errors when validating or sweeping
-k[ill]	Drop shadows	Drops unavailable shadows. NOTE: syntx is gfix -k (no database name)
–l[**ist**]	Transaction recovery	Displays IDs of each limbo transaction and indicates what would occur if **–t** were used for automated two-phase recovery
–m[**end**]	Data repair	Marks corrupt records as unavailable, so they are skipped (for example, during a subsequent backup)

TABLE 6.4 **gfix** options

Option	Task	Description
-mo[de] [read_write | read_only}	Set access mode	• Sets mode of database to either read-only or read-write • Default table mode is read_write • Requires exclusive access to the database
–n[**o_update**]	Data repair	Used with **–v** to validate corrupt or misallocated structures; structures are reported but not fixed
–o[**nline**]	Shutdown	Cancels a **–shut** operation that is scheduled to take effect or rescinds a shutdown that is currently in effect
–pa[**ssword**] *text*	Remote access	Checks for password *text* before accessing a database
–p[**rompt**]	Transaction recovery	Used with **–l** to prompt for action during transaction recovery
–r[**ollback**] **{ID | all}**	Transaction recovery	Rolls back limbo transaction specified by **ID** or roll back all limbo transactions
–s[**weep**]	Sweeping	Forces an immediate sweep of the database • Useful if automatic sweeping is disabled • Exclusive access is not necessary
-s[ql_dialect] *n*	Database dialect	Changes database dialect to *n* • Dialect 1 = InterBase 5.*x* compatibility • Dialect 3 = InterBase 6 with new SQL92 features
–sh[**ut**]	Shutdown	• Shuts down the database • Must be used in conjunction with **–attach**, **–force**, or **–tran**
–t[**wo_phase**] **{ID | all}**	Transaction recovery	Performs automated two-phase recovery, either for a limbo transaction specified by **ID** or for all limbo transactions
–tr[**an**] *n*	Shutdown	Used with **–shut** to prevent new transactions from starting during timeout period of *n* seconds; cancels shutdown if there are still active transactions after *n* seconds
–user *name*	Remote access	Checks for user *name* before accessing a remote database

TABLE 6.4 **gfix** options (*continued*)

Option	Task	Description
–v[**alidate**]	Data repair	Locates and releases pages that are allocated but unassigned to any data structures; also reports corrupt structures
–w[**rite**] **{sync | async}**	Database writes	Enables or disables forced (synchronous) writes **sync** enables forced writes; **async** enables buffered writes
–z		Shows version of **gfix** and of the InterBase engine

TABLE 6.4 **gfix** options (*continued*)

Examples The following example changes the dialect of the **customer.gdb** database to 3:

```
gfix -sql 3 customer.gdb
```

The following example changes the customer.gdb database to read-only mode:

```
gfix -mo read-only customer.gdb
```

gfix error messages

Error Message	Causes and Suggested Actions to Take
Database file name <*string*> already given	A command-line option was interpreted as a database file because the option was not preceded by a hyphen (-) or slash (/). Correct the syntax.
Invalid switch	A command-line option was not recognized.
Incompatible switch combinations	You specified at least two options that do not work together, or you specified an option that has no meaning without another option (for example, **-full** by itself).
More limbo transactions than fit. Try again.	The database contains more limbo transactions than **gfix** can print in a single session. Commit or roll back some of the limbo transactions, then try again.
Numeric value required	The **-housekeeping** option requires a single, non-negative argument specifying number of transactions per sweep.
Please retry, specifying <*string*>	Both a file name and at least one option must be specified.

TABLE 6.5 **gfix** database maintenance error messages

Error Message	**Causes and Suggested Actions to Take**
Transaction number or “all” required	You specified **-commit**, **-rollback**, or **-two_phase** without supplying the required argument.
-mode read_only or read_write	The **-mode** option takes either read_only or read_write as an option.
“read_only” or “read_write” required	The **-mode** option must be accompanied by one of these two arguments.

TABLE 6.5 **gfix** database maintenance error messages (*continued*)

CHAPTER

7

Database Backup and Restore

A database *backup* saves a database to a file on a hard disk or other storage medium. To protect a database from power failure, disk crashes, or other potential data loss, you should regularly back up the database. For additional safety, it is recommended to store the backup medium in a different physical location from the database server.

A database *restore* re-creates a database from a backup file.

Benefits of backup and restore

Operating systems usually include facilities to archive database files. Using the InterBase backup and restore feature in **gbak** or IBConsole offers several advantages over such backup methods. The backup and restore process accomplishes the following:

- Improves database performance, by performing garbage collection on outdated records, and balances indexes.
- Reclaims space occupied by deleted records, and pack the remaining data. This often reduces database size.
- Gives you the option of changing the database page size or distributing the database among multiple files or disks.

- Enables backups to run concurrently while other users are using the database. You do not have to shut down the database to run a back up. However, any data changes that clients commit to the database after the backup begins are not recorded in the backup file.
- Provides you with a platform-independent, stable snapshot of the database for archiving purposes.
- Creates a database backup to a disk file or to a named tape device.
- Upgrades the ODS

  New major releases of the InterBase server often contain changes to the on-disk structure (ODS). If the ODS has changed and you want to take advantage of any new InterBase features, upgrade your databases to the new ODS.

  You need not upgrade databases to use a new version of InterBase. The new versions can still access databases created with a previous version, but cannot take advantage of any new InterBase features.

  To upgrade existing databases to a new ODS, perform the following steps:

  1. Before installing the new version of InterBase, back up databases using the old version.
  2. Install the new version of the InterBase server as described in *Getting Started*.
  3. Once the new version is installed, restore the databases with the new version of InterBase.

  The restored databases are able to use any new InterBase server features.

Database ownership

Although backing up a database can be performed by only the owner or SYSDBA, any user can restore a database as long as they are not restoring it over an existing database. A restored database file belongs to the user ID of the person who performed the restore. This means that backing up and restoring a database is a mechanism for changing the ownership of a database. It also means that an unauthorized user can “steal” a database by restoring a backup file to a machine where he knows the SYSDBA password. It is important to ensure that your backup files are secured from unauthorized access.

Note To restore a database over an existing database, you must be the owner of the existing database or SYSDBA.

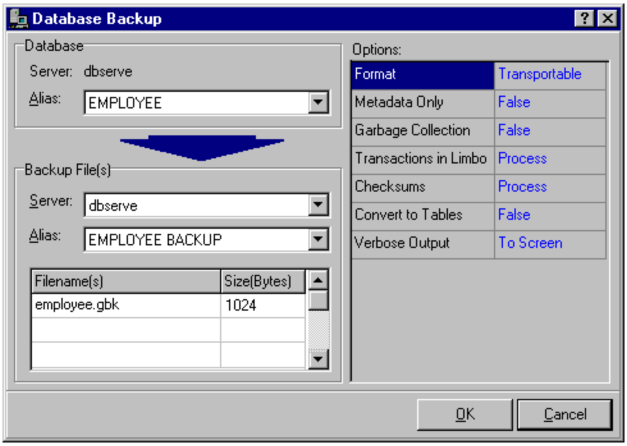
Backing up a database using IBConsole

Use the Database Backup dialog to back up a database. To access this dialog, select a logged-in server from the list of available servers displayed in the Tree pane and continue with any of the following methods:

- Select Databases or any database under the Databases hierarchy and choose **Database | Maintenance | Backup/Restore | Backup**.
- Right-click any connected database under the Databases hierarchy and choose **Backup/Restore | Backup** from the context menu.
- Select a connected database under the Databases hierarchy and choose Database Backup in the Work pane.
- Select a database alias under Backup in the Tree pane and choose Backup in the Work pane.

The Database Backup dialog appears:

FIGURE 7.1 Database backup dialog



Database

Server: dbserve

Alias: EMPLOYEE

Backup File(s)

Server: dbserve

Alias: EMPLOYEE BACKUP

Filename(s)	Size(Bytes)
employee.gbk	1024

Options:

Format	Transportable
Metadata Only	False
Garbage Collection	False
Transactions in Limbo	Process
Checksums	Process
Convert to Tables	False
Verbose Output	To Screen

To back up a database:

1. Check the database server to make sure the server indicated is correct. If it is not, cancel this dialog and re-initiate the Database Backup dialog under the correct server.
2. If you accessed the Database Backup dialog from a database alias then the Alias field is automatically assigned. If, however, you accessed the Database Backup dialog from Databases, then you must select an alias from the list of database aliases.

   Note The database alias references the necessary database files information associated with the database so you only need to specify the alias name, not the actual database filename, when indicating the database to backup. If the database spans multiple files, the server uses the header page of each file to locate additional files, so the entire database can be backed up based on the alias filename.
3. Select a destination server from a list of registered servers in the Backup Files Server drop down list.
4. Once a destination server has been selected a list of backup file aliases is available from the Backup Files Alias drop down list. If you want to overwrite an existing backup file, select the appropriate file from the drop down list. If you want to create a new backup file you can type a new alias name in the Backup File(s) Alias field.
5. Next you must indicate where the backup is to be stored by entering one or more filenames, specifying a size for each file, in the Backup File(s) table. To insert a new row into the Backup File(s) table, move to the last row and column in the table and type Ctrl-Tab.

   When entering a filename, make sure to include the file path unless you wish to default the file to the working directory.

   If you select an existing backup alias the table displays all the filenames and file sizes of that alias. You can edit any information within this table. You can add another file to the backup file list by entering a new filename at the end of the table. You can remove a file from the backup file list by deleting (blanking out) the values in the table.
6. You can specify backup options by entering a valid value, by clicking the option value and choosing a new value from a drop down list of values or by double-clicking the option value to rotate its value to the next in the list of values. See **“Backup options”** below for descriptions of these options.
7. Click OK to start the backup.

Note Database files and backup files can have any name that is legal on the operating system; the **.gdb** and **.gbk** file extensions are InterBase conventions only.

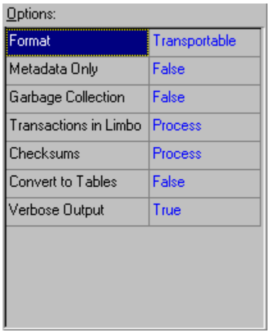
A backup file typically occupies less space than the database because it includes only the current version of data and incurs less overhead for data storage. A backup file also does not contain index data structures, only the index definition.

If you specify a backup file that already exists, IBConsole overwrites it. To avoid overwriting, specify a unique name for the backup file.

Backup options

The backup options are shown on the right side of the Database Backup dialog. You can specify options by entering a value, by clicking the option value and choosing a new value from a drop down list of values or by double-clicking the option value to rotate its value to the next in the list of values.

FIGURE 7.2 Database backup options



Options:

Format	Transportable
Metadata Only	False
Garbage Collection	False
Transactions in Limbo	Process
Checksums	Process
Convert to Tables	False
Verbose Output	True

Format

Option values are Transportable and Non-transportable.

To move a database to a machine with a different operating system from the machine on which the backup was performed, make sure the Format option is set to Transportable. This option writes data in a generic format, enabling you to move to any machine that supports InterBase.

Note Never copy a database from one location to another. Back it up and then restore it to the new location.

Metadata Only

Option values are True and False.

When backing up a database, you can exclude its data, saving only its metadata. You might want to do this to:

- Retain a record of the metadata before it is modified.
- Create an empty copy of the database. The copy has the same metadata but can be populated with different data.

To back up metadata only, select True for the Metadata Only option.

Tip You can also extract a database's metadata using **isql**. **isql** produces an SQL data definition (text) file containing SQL commands. IBConsole backup Metadata Only creates a binary backup file containing only metadata.

This function corresponds to the **-metadata** option of **gbak.**

Garbage Collection

Option values are True and False.

By default, IBConsole performs garbage collection during backup. To prevent garbage collection during a backup, set the Garbage Collection option value to False.

Garbage collection marks space used by old versions of data records as free for reuse. Generally, you want IBConsole to perform garbage collection during backup.

Tip You do not want to perform garbage collection if there is data corruption in old record versions and you want to prevent InterBase from visiting those records during a backup.

This function corresponds to the **-garbage_collect** option of **gbak**.

Transactions in Limbo

Option values are Process and Ignore.

To ignore limbo transactions during backup, set the Transactions in Limbo option value to Ignore.

When IBConsole ignores limbo transactions during backup, it ignores all record versions created by any limbo transaction, finds the most recently committed version of a record, and backs up that version.

Limbo transactions are usually caused by the failure of a two-phase commit. They can also exist due to system failure or when a single-database transaction is prepared.

Before backing up a database that contains limbo transactions, it is a good idea to perform transaction recovery, by choosing **Database | Maintenance | Transaction Recovery** in the Database Maintenance window. Refer to **"Recovering transactions" on page 135** for more information.

This function corresponds to the **-limbo** option of **gbak**.

▸ *Checksums*

Option values are Process and Ignore.

To ignore checksums during backup, set the Checksums option value to Ignore.

A checksum is a page-by-page analysis of data to verify its integrity. A bad checksum means that a data page has been randomly overwritten; for example, due to a system crash.

Checksum errors indicate data corruption, and InterBase normally prevents you from backing up a database if bad checksums are detected. Examine the data the next time you restore the database.

This function corresponds to the **-ignore** option of **gbak**.

▸ *Convert to Tables*

To convert external files to internal tables, set the Convert to Tables option value to True.

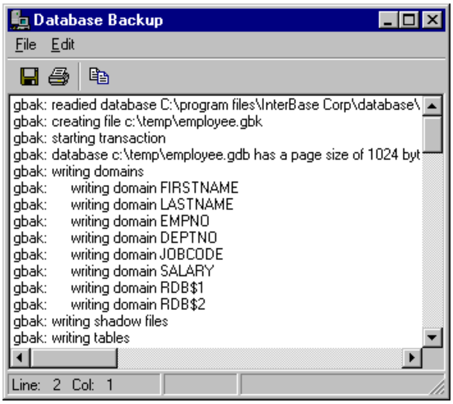
This function corresponds to the **-convert** option of **gbak**.

▸ *Verbose Output*

Option values are None, To Screen and To File.

To monitor the backup process as it runs, set the Verbose Output option value to To Screen. This option opens a standard text display window to display status messages during the backup. For example:

FIGURE 7.3 Database backup verbose output



```
gbak: readied database C:\program files\InterBase Corp\database\
gbak: creating file c:\temp\employee.gbk
gbak: starting transaction
gbak: database c:\temp\employee.gdb has a page size of 1024 byt
gbak: writing domains
gbak:     writing domain FIRSTNAME
gbak:     writing domain LASTNAME
gbak:     writing domain EMPNO
gbak:     writing domain DEPTNO
gbak:     writing domain JOBCODE
gbak:     writing domain SALARY
gbak:     writing domain RDB$1
gbak:     writing domain RDB$2
gbak: writing shadow files
gbak: writing tables
```

The standard text display window enables you to search for specific text, save the text to a file, and print the text. For an explanation of how to use the standard text display window, see **"Standard text display window" on page 37.**

This function corresponds to the **-verbose** option of **gbak.**

Transferring databases to servers running other operating systems

1. Set the Format option to Transportable in the Database Backup dialog.
2. Back up the database.
3. If you backed up to a removable medium, proceed to Step 4. If you created a backup file on disk, use operating-system commands to copy the file to a removable medium, such as a tape. Then load the contents of the medium onto another machine, or copy it across a network to another machine.
4. On the destination machine, restore the backup file. If restoring from a removable medium, such as tape, specify the device name instead of the backup file.

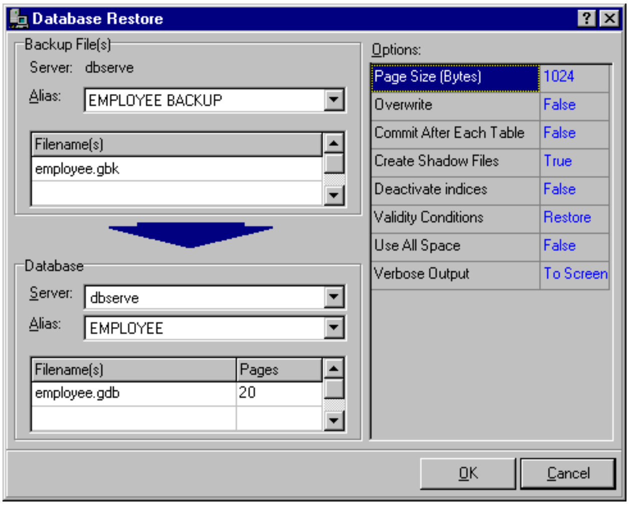
Restoring a database using IBConsole

Use the Database Restore dialog to restore databases. To access this dialog, select a server from the list of available servers displayed in the Tree pane and continue with one of these possible methods:

- Select anything under the databases hierarchy and choose **Database | Maintenance | Backup/Restore | Restore**.
- Double-click any backup alias name under the Backup hierarchy.
- Right-click Backup or any backup alias name under the Backup hierarchy and choose Restore from the context menu.
- Select any backup alias name under Backup and click Restore in the Work pane.

The Database Restore dialog appears:

FIGURE 7.4 Database Restore dialog



Database Restore

Backup File(s)

Server: dbserve

Alias: EMPLOYEE BACKUP

Filename(s)
employee.gbk

Database

Server: dbserve

Alias: EMPLOYEE

Filename(s)	Pages
employee.gdb	20

Options:

Page Size (Bytes)	1024
Overwrite	False
Commit After Each Table	False
Create Shadow Files	True
Deactivate indices	False
Validity Conditions	Restore
Use All Space	False
Verbose Output	To Screen

IMPORTANT When restoring a database, do not replace a database that is currently in use.

To restore a database:

1. Check the source Backup File(s) Server to make sure the server indicated is correct. If it is not, cancel this dialog and re-initiate the Database Restore dialog under the correct server.
2. If you accessed the Database Restore dialog from a backup alias, then the Alias field is automatically assigned. If, however, you accessed the Database Restore dialog from Backup, then you must select an alias from the list of backup aliases.

Note The backup alias references the necessary database backup files information associated with the database so you only need to specify the alias name, not the actual backup filename, when indicating the backup to restore. If the backup spans multiple files, the server uses header page of each file to locate additional files, so the entire backup can be restored based on the alias filename.

3. If you choose a backup file alias, the Backup File(s) table displays the associated backup files. If you do not specify a backup file alias, then you can enter the backup filenames manually, or you can browse for the file by selecting "File..." from the Alias drop-down list. When entering a filename, be sure to include the file path where the file is located. It is important that you include all filenames associated with the restore. To insert a new row into the Backup File(s) table, move to the last row and column in the table and type Ctrl-Tab.
4. Select a destination server from a list of registered servers in the Database Server drop down list.
5. Once a destination server has been selected a list of database aliases is available from the Database Alias drop down list. If you want to restore to an existing database, select the appropriate alias from the drop down list. If you want to restore to a new database, type a new alias name in the Database Alias field.
6. Next you must indicate where the backup is to be restored to by entering one or more filenames, specifying the number of pages required for each file, in the Database table. When entering a filename, make sure to include the file path unless you wish to default the file to the working directory. To insert a new row into the Database table, move to the last row and column in the table and type Ctrl-Tab.

You might want to restore a database to multiple files to distribute it among different disks, which provides more flexibility in allocating system resources.

You can add another file to the backup file list by entering a new filename at the end of the table. You can remove a file from the backup file list by deleting (blanking out) the values in the table.

If you selected an existing database alias the Database table will display all the associated filenames and number of pages. You can edit any information within this table. You can add another file to the database file list by entering a new filename at the end of the table. You can remove a file from the list by deleting (blanking out) the values in the table.

Note You cannot restore a database to a network file system (mapped drive).

7. You can specify restore options by entering a valid value, by clicking the option value and choosing a new value from a drop down list of values or by double-clicking the option value to rotate its value to the next in the list of values. See **“Restore options”** below for a description of these options.

8. Click OK to start the restore.

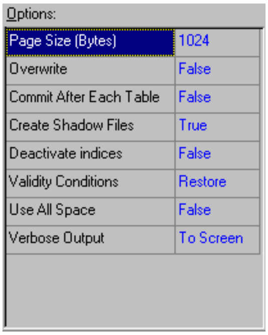
   Typically, a restored database occupies less disk space than it did before being backed up, but disk space requirements could change if the on-disk structure version changes. For information about the ODS, see **“Benefits of backup and restore” on page 143**.

Note InterBase 6 restore allows you to restore a database successfully even if for some reason the restore process could not rebuild indexes for the database. For example, this can occur if there is not enough temporary disk space to perform the sorting task necessary to build an index during the restore. If this occurs, the database is restored and available, but indexes are inactive. This is as if you had set the Deactivate Indexes option to True, or used the **-i** switch of **gbak**. After the restore completes, use ALTER INDEX to set the indexes active.

Restore options

The restore options are shown on the right side of the Database Restore dialog. You can specify options by entering a value, by clicking the option value and choosing a new value from a drop down list of values or by double-clicking the option value to rotate its value to the next in the list of values.

FIGURE 7.5 Database restore options



Options:

Page Size (Bytes)	1024
Overwrite	False
Commit After Each Table	False
Create Shadow Files	True
Deactivate indices	False
Validity Conditions	Restore
Use All Space	False
Verbose Output	To Screen

Page Size

InterBase supports database page sizes of 1024, 2048, 4096, and 8192 bytes. The default is 4096 bytes. To change page size, back up the database and then restore it, modifying the Page Size option in the Database Restore dialog.

Changing the page size can improve performance for the following reasons:

- Storing and retrieving Blob data is most efficient when the entire Blob fits on a single database page. If an application stores many Blobs exceeding 4KB, using a larger page size reduces the time for accessing Blob data.
- InterBase performs better if rows do not span pages. If a database contains long rows of data, consider increasing the page size.
- If a database has a large index, increasing the database page size reduces the number of levels in the index tree. Indexes work faster if their depth is kept to a minimum. Choose **Database | Maintenance | Database Statistics** to display index statistics, and consider increasing the page size if index depth is greater than three on any frequently used index.
- If most transactions involve only a few rows of data, a smaller page size may be appropriate, because less data needs to be passed back and forth and less memory is used by the disk cache.

This function corresponds to the -**page_size** option of **gbak**.

Overwrite

Option values are True and False.

IBConsole cannot overwrite an existing database file unless the Overwrite option value is set to True. If you attempt to restore to an existing database name and this option is set to False, the restore does not proceed.

Note To restore a database over an existing database, you must be the owner of the existing database or SYSDBA.

IMPORTANT Do not replace an existing database while clients are operating on it. When restoring to an existing file name, a safer approach is to rename the existing database file, restore the database, then drop or archive the old database as needed.

This function corresponds to the -**replace** option of **gbak**.

Commit After Each Table

Option values are True and False.

Normally, IBConsole restores all metadata before restoring any data. If you set the Commit After Each Table option value to True, IBConsole restores the metadata and data for each table together, committing one table at a time.

This option is useful when you are having trouble restoring a backup file; for example, if the data is corrupt or invalid according to integrity constraints.

If you have a problem backup file, restoring the database one table at a time lets you recover some of the data intact. You can restore only the tables that precede the bad data; restoration fails the moment it encounters bad data.

This function corresponds to the -**one_at_a_time** option of **gbak**.

Create Shadow Files

Shadow files are identical, physical copies of database files in a database. To recreate shadow files that were saved during the backup process set the Create Shadow Files option to True. For further information on shadowing see **"Shadowing" on page 111**.

Deactivate Indexes

Option values are True and False.

Normally, InterBase rebuilds indexes when a database is restored. If the database contained duplicate values in a unique index when it was backed up, restoration fails. Duplicate values can be introduced into a database if indexes were temporarily made inactive (for example, to allow insertion of many records or to rebalance an index).

To enable restoration to succeed in this case, set the Deactivate Indexes option to True. This makes indexes inactive and prevents them from rebuilding. Then eliminate the duplicate index values, and re-activate indexes through ALTER INDEX in **isql**.

A unique index cannot be activated using the ALTER INDEX statement; a unique index must be dropped and then created again. For more information about activating indexes, see the *Language Reference*.

Tip The Deactivate Indexes option is also useful for bringing a database online more quickly. Data access is slower until indexes are rebuilt, but the database is available. After the database is restored, users can access it while indexes are reactivated.

This function corresponds to the -**inactive** option of **gbak**.

Validity Conditions

Option values are Restore and Ignore.

If you redefine validity constraints in a database where data is already entered, your data might no longer satisfy the validity constraints. You might not discover this until you try to restore the database, at which time an error message about invalid data appears.

IMPORTANT Always make a copy of metadata before redefining it; for example, by extracting it using **isql**.

To restore a database that contains invalid data, set the Validity Conditions option to Ignore. This option deletes validity constraints from the metadata. After the database is restored, change the data to make it valid according to the new integrity constraints. Then add back the constraints that were deleted.

This option is also useful if you plan to redefine the validity conditions after restoring the database. If you do so, thoroughly test the data after redefining any validity constraints.

This function corresponds to the -**no_validity** option of **gbak**.

▸ *Use All Space*

Option values are True and False.

To restore a database with 100% fill ratio on every data page, instead of the default 80% fill ratio, set the Use All Space option to True.

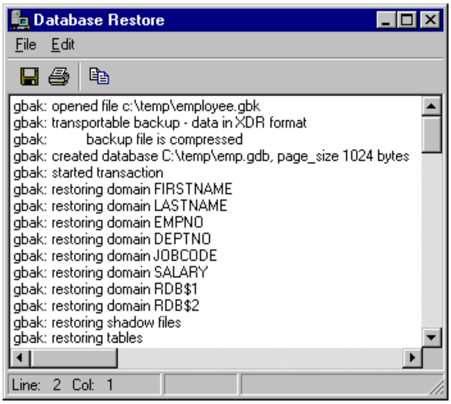
This function corresponds to the **-use_all_space** option of **gbak**.

▸ *Verbose Output*

Option values are None, To Screen, and To File.

To monitor the restore process as it runs, set the Verbose Output option to To Screen. This option opens a standard text display window to display status messages during the restore. For example:

FIGURE 7.6 Database restore verbose output



```
gbak: opened file c:\temp\employee.gbk
gbak: transportable backup - data in XDR format
gbak:         backup file is compressed
gbak: created database C:\temp\emp.gdb, page_size 1024 bytes
gbak: started transaction
gbak: restoring domain FIRSTNAME
gbak: restoring domain LASTNAME
gbak: restoring domain EMPNO
gbak: restoring domain DEPTNO
gbak: restoring domain JOBCODE
gbak: restoring domain SALARY
gbak: restoring domain RDB$1
gbak: restoring domain RDB$2
gbak: restoring shadow files
gbak: restoring tables
```

The standard text display window enables you to search for specific text, save the text to a file, and print the text. For an explanation of how to use the standard text display window, see **"Standard text display window" on page 37.**

This function corresponds to the -**verbose** option of **gbak**.

gbak command-line tool

The **gbak** command-line tool allows both back up or restore of a database, with options for changing specified database characteristics. Only SYSDBA or database owner can back up a database.

Database backup

Syntax **For backing up to a single file:**

```
gbak [-B] [options] database target
```

When backing up from a multifile database, specify only the first file name.

For backing up to multiple files:

```
gbak [-B] [options] database target1 size1[k|m|g] target2
    [size2[k|m|g] target3
```

When backing up from a multifile database, specify only the first file name.

Argument	Description
database	• Name of a database to back up • For a multifile database, the name of the first database file
target	Name of a storage device or backup file to which to back up • On UNIX, can also be **stdout**, in which case **gbak** writes its output to the standard output (usually a pipe) • No size need be specified when restoring to a single file, since the database always expands as needed to fill all available space
size	Length of a backup file or restored database file • The only permissible unit for a restored database file is database pages; minimum value is 200 • Default unit for a backup file is bytes • Size of backup files can also be specified in kilobytes, megabytes, or gigabytes • Do not specify a size for the final backup file or database file; the last file always expands as needed to fill all available space

TABLE 7.1 **gbak** arguments

Options In the OPTION column of the following tables, only the characters outside the square brackets ([ ]) are required.

Table 7.2 lists the options to **gbak** that are available for creating backups.

Option	Description
-b[**ackup_database**]	Backs up database to file or device
-co[**nvert**]	Converts external files as internal tables
-e[**xpand**]	Creates a noncompressed back up
-fa[**ctor**] *n*	Uses blocking factor *n* for tape device
-g[**arbage_collect**]	Does not garbage collect during backup
-ig[**nore**]	Ignores checksums during backup
-l[**imbo**]	Ignores limbo transactions during backup
-m[**etadata**]	Backs up metadata only, no data

TABLE 7.2 **gbak** backup options

Option	Description
-nt	Creates the backup in nontransportable format
-ol[**d_descriptions**]	Backs up metadata in old-style format
-pa[**ssword**] *text*	Checks for password *text* before accessing a database
-role *name*	Connects as role *name*
-se[**rvice**] *servicename*	• Creates the backup files on the host where the original database files are located, using InterBase's Service Manager • *servicename* invokes the Service Manager on the server host; syntax varies with the network protocol in use: TCP/IP *hostname*:service_mgr SPX *hostname*@service_mgr Named pipes \\hostname\service_mgr Local service_mgr
-t[**ransportable**]	Creates a transportable backup [default]
-u[**ser**] *name*	Checks for user *name* before accessing remote database
-v[**erbose**]	Shows what **gbak** is doing
-y [*file* **|** **suppress_output**]	Direct status messages to *file*; *file* must not already exist; **suppress_output** suppress output messages
-z	Show version of **gbak** and of InterBase engine

TABLE 7.2 **gbak** backup options (*continued*)

Backing up a database with gbak

When backing up a database, bear the following points in mind:

- Only the database owner or SYSDBA can back up a database.

- Unless the **-service** option is specified, **gbak** writes the backup files to the current directory of the machine on which it is running, not on the server where the database resides. If you specify a location for the backup file, it is relative to the machine where **gbak** is executing. You can write the backup files only to this local machine or to drives that are mapped to it. Note that the **-service** switch changes this behavior. (See **"Using gbak with InterBase Service Manager" on page 163**.)
- When you are backing up a multifile database, specify *only* the first file in the backup command. You must not name the subsequent database files: they will be interpreted as backup file names.
- The default unit for backup files is bytes. You can choose to specify kilobytes, megabytes, or gigabytes (**k**, **m**, or **g**) instead. Restored database files can be specified only in database pages.

  Note It is good security practice to change your backup files to read-only at the system level after creating them. This prevents them from being accidentally overwritten. In addition, you can protect your databases from being "kidnapped" on UNIX and NT/2000 systems by placing the backup files in directories with restricted access.

Tip Use the **-transportable** switch if you operate in a multiplatform environment. This switch permits the database to be backed up to a platform other than the one on which it originally resided. Using this option routinely is a good idea when you are operating in a multiplatform environment.

Tip Use the **-service** switch if you are backing up to the same server that holds the original database. This option invokes the InterBase Service Manager on the server host and saves both time and network traffic.

Restoring a database with gbak

Syntax **For restoring:**

```
gbak {-C|-R} [options] source dbfile
```

For restoring to multiple files:

```
gbak {-C|-R} [options] source dbfile1 size1 dbfile2
   [size2 dbfile3 …]
```

For restoring from multiple files:

```
gbak {-C|-R} [options] source1 source2 [source3 …] dbfile
```

By extension, you can restore *from* multiple files *to* multiple files using the following syntax:

```
gbak {-C|-R} [options] source1 source2 [source3 …] dbfile1
   size1 dbfile2 [size2 dbfile3 …]
```

Argument	**Description**
source	Name of a storage device or backup file from which to restore On UNIX, this can also be **stdin**, in which case **gbak** reads input from the standard input (usually a pipe).
dbfile	The name of a restored database file
size	Length of a backup file or restored database file • The only permissible unit for a restored database file is database pages; minimum value is 200 • Default unit for a backup file is bytes • Size of backup files can also be specified in kilobytes, megabytes, or gigabytes • Do not specify a size for the final backup file or database file; the last file always expands as needed to fill all available space

Table 0-1:

Table 7.3 lists **gbak** options that are available when restoring databases.

Option	**Description**
-c[**reate_database**]	Restores database to a new file
-bu[**ffers**]	Sets cache size for restored database
-i[**nactive**]	Makes indexes inactive upon restore
-k[**ill**]	Does not create any shadows that were previously defined
-mo[de] [read_write | read_only}	Specifies whether the restored database is writable • Possible values are **read_only** and **read_write** • Default is **read_write**
-n[**o_validity**]	Deletes validity constraints from restored metadata; allows restoration of data that would otherwise not meet validity constraints

TABLE 7.3 **gbak** restore options

Option	Description
-o[**ne_at_a_time**]	Restores one table at a time; useful for partial recovery if database contains corrupt data
-p[**age_size**] *n*	Resets page size to *n* bytes (1024, 2048, 4096, or 8192); default is 4096
-pa[**ssword**] *text*	Checks for password *text* before accessing a database
-r[**eplace_database**]	Restores database to new file or replaces existing file
-se[**rvice**] *servicename*	• Creates the restored database on the host where the backup files are located, using InterBase's Service Manager • *servicename* invokes the Service Manager on the server host; syntax varies with the network protocol in use: TCP/IP *hostname*:service_mgr SPX *hostname*@service_mgr Named pipes \\hostname\\service_mgr
-user *name*	Checks for user *name* before accessing database
-use_[**all_space**]	Restores database with 100% fill ratio on every data page, instead of the default 80% fill ratio
-v[**erbose**]	Shows what **gbak** is doing
-y [*file* | **suppress_output**]	If used with **-v**, directs status messages to *file;* if used without **-v** and *file* is omitted, suppresses output messages
-z	Show version of **gbak** and of InterBase engine

TABLE 7.3 **gbak** restore options (*continued*)

When restoring a database, bear the following points in mind:

- Anyone can restore a database. However, only the database owner or SYSDBA can restore a database over an existing database.
- Do not restore a backup over a database that is currently in use; it is likely to corrupt the database.
- When restoring from a multifile backup, name all the backup files, in any order.

- Do not provide a file size for the last (or only) file of the restored database. InterBase does not return an error, but it always "grows" the last file as needed until all available space is used. This dynamic sizing is a feature of InterBase.
- You specify the size of a restored database in database pages. The default size for database files is 200 pages. The default database page size is 4K, so if the page size has not been changed, the default database size is 800K. This is sufficient for only a very small database. To change the size of the database pages, use the **-p**[**age_size**] option when restoring.

Tip Use the **-service** switch if you are restoring to the same server that holds the backup file. This option invokes the InterBase Service Manager on the server host and saves both time and network traffic.

Note If you specify several target database files but have only a small amount of data, the target files are quite small (around 800K for the first one and 4K for subsequent ones) when they are first created. They grow in sequence to the specified sizes as you populate the database.

Using gbak with InterBase Service Manager

When you run **gbak** with the **-service** switch, **gbak** invokes the backup and restore functions of InterBase's Service Manager on the server where the database resides. When run without the **-service** switch, **gbak** executes on the machine where it is invoked—typically a client—and writes the backup file on (or relative to) that machine. Using the **-service** switch to invoke the Service Manager saves a significant amount of time and network traffic when you want to create the backup on the same host on which the database resides. You have the option of specifying another machine as the target when using the -**service** switch, but the advantages of reduced time and network traffic are lost.

When you use the **-service** switch, you specify the host name followed by the string "service_mgr". The syntax you use for this varies with the network protocol you are using. Together, these components are referred to as "host_service" in the syntax statements that follow in this section.

Network protocol	Syntax
TCP/IP	*hostname*`:service_mgr`
SPX	*hostname*`@service_mgr`
Named pipes	\\hostname`\service_mgr`
Local	`service_mgr`

TABLE 7.4 *host_service* syntax for calling the Service Manager with **gbak**

The syntax in the right column appears in the **gbak** syntax below as "host_service."

The "local" case is trivial on NT. If you are backing up a local database, the results in terms of time and network traffic are the same whether you use the **-service** switch or not, even though the actual implementation would be slightly different. On UNIX systems, the local case is equivalent to specifying (for TCP/IP) **localhost:service_mgr** and saves both time and network traffic.

Syntax **Backing up with Service Manager**

```
gbak -b [options] -se[rvice] host_service database filename
```

Syntax **Restoring with Service Manager**

```
gbak {-c|-r} [options] -se[rvice] host_service filename database
```

You can back up to multiple files and restore from multiple files using Service Manager.

IMPORTANT On UNIX systems, in order to restore a database that has been backed up using the Service Manager, you must either use the Service Manager for the restore or you must be logged onto the system as the user that InterBase was running as when the backup was created (either *root* or *interbase*). This is because the InterBase user (*root* or *interbase*) is the owner of the backup file at the system level when the Service Manager is invoked, and the backup file is readable to only that user. When **gbak** is used to back up a database without the **-service** option, the owner of the backup file at the system level is the login of the person who ran **gbak**. On Windows platforms, the system-level constraints do not apply.

The user name and password

When InterBase checks to see whether the user running **gbak** is authorized to do so, it determines the user according to the following hierarchy:

- The **-user** that is specified, with a correct password, as part of the **gbak** command

- The user and password specified in the ISC_USER and ISC_PASSWORD environment variables, provided they also exist in **isc4.gdb** (setting these environment variables is strongly *not* recommended, since it is extremely insecure)
- UNIX only: If no user is specified at any of the previous levels, InterBase uses the UNIX login if the user is running on the server or on a trusted host.

Some backup and restore examples

Note The following examples use forward slashes exclusively. InterBase accepts either forward or backward slashes for paths on Wintel platforms.

Backup examples

The following example backs up **foo.gdb**, which resides on the server **jupiter** and writes the backup file to the current directory of the client machine where **gbak** is running. foo.gdb can be either a single-file database or the name of the first file in a multifile database. Using this syntax (without the **-se** switch) copies a lot of data over the net.

```
gbak -b -user joe -pa blurf@ jupiter:/foo.gdb foo.gbk
```

The next example backs up **foo.gdb**, which resides on the server **jupiter** and writes the backup file to the **C:/archive** directory on the client machine where **gbak** is running. As before, **foo.gdb** can be a single file database or the name of the first file in a multifile database. This syntax causes the same amount of network traffic as the first example.

```
gbak -b -user joe -pa blurf@ jupiter:/foo.gdb C:\archive\foo.gbk
```

The next example backs up the same database on **jupiter**, but uses the **-se**[**rvice**] switch to invoke the Service Manager on **jupiter**, which writes the backup to the **\backup** directory on **jupiter**. This command causes very little network traffic and is therefore faster than performing the same task without the **-se** (**-service**) switch. Note that the syntax (`jupiter:service_mgr`) indicates a TCP/IP connection.

```
gbak -b -user joe -pa blurf@ -se jupiter:service_mgr
   /foo.gdb /backup/foo.gbk
```

The next example again backs up **foo1.gdb** on server **jupiter** to multiple files in the **/backup** directory on **jupiter** using the Service Manager. This syntax backs up a single file or multifile database and uses a minimum of time and network traffic. It converts external files as internal tables and creates a backup in a transportable format that can be restored on any InterBase-supported platform. To back up a multifile database, name only the first file in the backup command. In this example, the first two backup files are limited to 500K. The last one expands as necessary.

```
gbak -b -user joe -pa blurf@ -co -t -se jupiter:service_mgr
   /foo1.gdb /backup/backup1.gbk 500k /backup/backup2.gbk 500k
   /backup/lastBackup.gbk
```

Database restore examples

The first example restores a database that resides in the /**archive** directory on the machine where **gbak** is running and restores it to **jupiter**, overwriting an existing (but inactive) database.

```
gbak -r -user joe -pa blurf@ C:\archive\foo.gbk jupiter:/foo.gdb
```

The next example restores a multifile database from the **/backup** directory of jupiter to the **/companydb** directory of **jupiter**. This command runs on the server by invoking Service Manager, thus saving time and network traffic. In this example, the first two files of the restored database are 500 pages long and the last file grows as needed.

```
gbak -r user -joe -pa blurf@ -se jupiter:service_mgr /backup/foo1.gbk
   /backup/foo2.gbk /backup/fooLast.gbk /companydb/foo1.gdb 500
   /companydb/foo2.gdb 500 /companydb/fooLast.gdb
```

The next example executes on server Jupiter using Service Manager and restores a backup that is on Jupiter to another server called Pluto.

```
gbak -r user -joe -pa blurf@ -se jupiter:service_mgr
   /backup/foo.gbk pluto:/companydb/foo.gdb
```

gbak error messages

Error Message	Causes and Suggested Actions to Take
Array dimension for column <string> is invalid	Fix the array definition before backing up
Bad attribute for RDB$CHARACTER_SETS	An incompatible character set is in use
Bad attribute for RDB$COLLATIONS	Fix the attribute in the named system table
Bad attribute for table constraint	Check integrity constraints; if restoring, consider using the **-no_validity** option to delete validity constraints
Blocking factor parameter missing	Supply a numeric argument for “factor” option
Cannot commit files	• Database contains corruption or metadata violates integrity constraints • Try restoring tables using **-one_at_a_time** option, or delete validity constraints using **-no_validity** option
Cannot commit index <string>	• Data might conflict with defined indexes • Try restoring using “inactive” option to prevent rebuilding indexes
Cannot find column for Blob	
Cannot find table <string>	
Cannot open backup file <string>	Correct the file name you supplied and try again
Cannot open status and error output file <string>	• Messages are being redirected to invalid file name • Check format of file or access permissions on the directory of output file
Commit failed on table <string>	• Data corruption or violation of integrity constraint in the specified table • Check metadata or restore “one table at a time”
Conflicting switches for backup/restore	A backup-only option and restore-only option were used in the same operation; fix the command and execute again
Could not open file name <string>	Fix the file name and re-execute command
Could not read from file <string>	Fix the file name and re-execute command

TABLE 7.5 **gbak** backup and restore error messages

Error Message	Causes and Suggested Actions to Take
Could not write to file <string>	Fix the file name and re-execute command
Datatype n not understood	An illegal datatype is being specified
Database format n is too old to restore to	• The **gbak** version used is incompatible with the InterBase version of the database • Try backing up the database using the **-expand** or **-old** options and then restoring it
Database <string> already exists. To replace it, use the –R switch	• You used **-create** in restoring a back up file, but the target database already exists • Either rename the target database or use **-replace**
Could not drop database <string> (database might be in use).	• You used **-replace** in restoring a file to an existing database, but the database is in use • Either rename the target database or wait until it is not in use
Device type not specified	The **-device** option (Apollo only) must be followed by **ct** or **mt**; obsolete as of InterBase V3.3
Device type <string> not known	The **-device** option (Apollo only) was used incorrectly; obsolete as of InterBase V3.3
Do not recognize record type n	
Do not recognize <string> attribute n -- continuing	
Do not understand BLOB INFO item n	
Error accessing BLOB column <string> -- continuing	
ERROR: Backup incomplete	• The backup cannot be written to the target device or file system • Either there is insufficient space, a hardware write problem, or data corruption
Error committing metadata for table <string>	• A table within the database could be corrupt. • If restoring a database, try using **-one_at_a_time** to isolate the table

TABLE 7.5 **gbak** backup and restore error messages (*continued*)

Error Message	Causes and Suggested Actions to Take
Exiting before completion due to errors	• This message accompanies other error messages and indicates that back up or restore could not execute • Check other error messages for the cause.
Expected array dimension n but instead found m	Try redefining the problem array
Expected array version number n but instead found m	Try redefining the problem array
Expected backup database <string>, found <string>	Check the name of the backup file being restored
Expected backup description record	
Expected backup start time <string>, found <string>	
Expected backup version 1, 2, or 3. Found n	
Expected blocking factor, encountered <string>	The **-factor** option requires a numeric argument
Expected data attribute	
Expected database description record	
Expected number of bytes to be skipped, encountered <string>	
Expected page size, encountered <string>	The **-page_size** option requires a numeric argument
Expected record length	
Expected volume number n, found volume n	When backing up or restoring with multiple tapes, be sure to specify the correct volume number
Expected XDR record length	
Failed in put_blr_gen_id	
Failed in store_blr_gen_id	
Failed to create database <string>	The target database specified is invalid; it might already exist
column <string> used in index <string> seems to have vanished	• An index references a non-existent column • Check either the index definition or column definition
Found unknown switch	An unrecognized **gbak** option was specified

TABLE 7.5 **gbak** backup and restore error messages (*continued*)

Error Message	Causes and Suggested Actions to Take
Index <string> omitted because n of the expected m keys were found	
Input and output have the same name. Disallowed.	A backup file and database must have unique names; correct the names and try again
Length given for initial file (n) is less than minimum (m)	• In restoring a database into multiple files, the primary file was not allocated sufficient space • InterBase automatically increases the page length to the minimum value • No action necessary
Missing parameter for the number of bytes to be skipped	
Multiple sources or destinations specified	Only one device name can be specified as a source or target
No table name for data	• The database contains data that is not assigned to any table • Use **gfix** to validate or mend the database
Page size is allowed only on restore or create	The **-page_size** option was used during a back up instead of a restore
Page size parameter missing	The **-page_size** option requires a numeric argument
Page size specified (n bytes) rounded up to m bytes	Invalid page sizes are rounded up to 1024, 2048, 4096, or 8192, whichever is closest
Page size specified (n) greater than limit (8192 bytes)	Specify a page size of 1024, 2048, 4096, or 8192
Password parameter missing	• The back up or restore is accessing a remote machine • Use **-password** and specify a password
Protection is not there yet	Unimplemented option **-unprotected** used
Redirect location for output is not specified	You specified an option reserved for future use by InterBase
REPLACE specified, but the first file <string> is a database	Check that the file name following the **-replace** option is a backup file rather than a database
Requires both input and output file names	Specify both a source and target when backing up or restoring

TABLE 7.5 **gbak** backup and restore error messages (*continued*)

Error Message	Causes and Suggested Actions to Take
RESTORE: decompression length error	• Possible incompatibility in the **gbak** version used for backing up and the **gbak** version used for restoring • Check whether **-expand** should be specified during back up
Restore failed for record in table <string>	Possible data corruption in the named table
Skipped n bytes after reading a bad attribute n	
Skipped n bytes looking for next valid attribute, encountered attribute m	
Trigger <string> is invalid	
Unexpected end of file on backup file	• Restoration of the backup file failed; the backup procedure that created the backup file might have terminated abnormally • If possible, create a new backup file and use it to restore the database
Unexpected I/O error while <string> backup file	A disk error or other hardware error might have occurred during a backup or restore
Unknown switch <string>	An unrecognized **gbak** option was specified
User name parameter missing	• The backup or restore is accessing a remote machine • Supply a user name with the **–user** option
Validation error on column in table <string>	• The database cannot be restored because it contains data that violates integrity constraints • Try deleting constraints from the metadata by specifying **–no_validity** during restore
Warning -- record could not be restored	Possible corruption of the named data
Wrong length record, expected n encountered n	

TABLE 7.5 **gbak** backup and restore error messages (*continued*)

CHAPTER

8

Database and Server Statistics

This chapter provides an overview of the following InterBase facilities:

- Viewing a database summary and analysis
- Using the **gstat** command-line tool
- Viewing lock statistics
- Using the **iblockpr** command-line tool
- Retrieving statistics programmatically

Viewing statistics using IBConsole

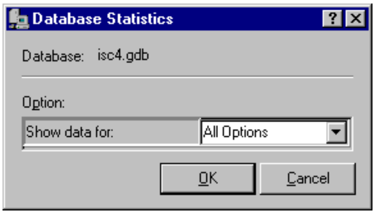
To view database statistics, use one of the following methods to access the Database Statistics dialog:

- Select a connected database in the Tree pane and choose **Database | Maintenance | Database Statistics**.
- Select a connected database in the Tree pane and click Database Statistics in the Work pane.

- Right-click a connected database in the Tree pane and choose **Maintenance | Database Statistics** from the context menu.

  A Database Statistics dialog appears where you can select which statistics you want to display.

FIGURE 8.1 Database Statistics options



Database: isc4.gdb

Option:

Show data for: All Options

To view database statistics:

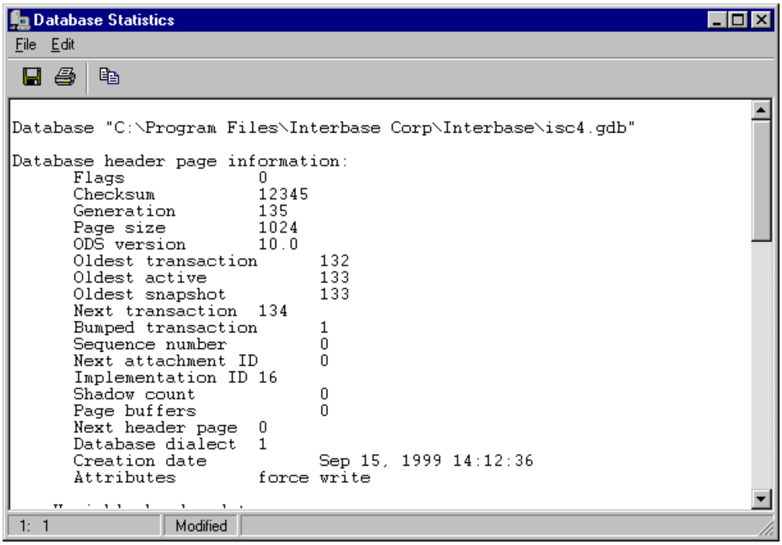
1. Select the statistical data you wish to generate from the Options list.

   You can specify options by entering a value, by clicking the option value and choosing a new value from a drop down list of values or by double-clicking the option value to rotate its value to the next in the list of values.

2. Click OK to generate database statistics.

Note In some cases, it can take a long time to display the statistics for large databases because, depending on what information has been selected to report, generating these statistics may analyze all the tables and indexes in a database.

FIGURE 8.2 Database Statistics dialog



```
Database "C:\Program Files\Interbase Corp\Interbase\isc4.gdb"

Database header page information:
      Flags             0
      Checksum          12345
      Generation        135
      Page size         1024
      ODS version       10.0
      Oldest transaction      132
      Oldest active           133
      Oldest snapshot         133
      Next transaction  134
      Bumped transaction      1
      Sequence number         0
      Next attachment ID      0
      Implementation ID 16
      Shadow count            0
      Page buffers            0
      Next header page  0
      Database dialect  1
      Creation date           Sep 15, 1999 14:12:36
      Attributes        force write
```

The Database Statistics report dialog is a standard text display window that exhibits database summary and database analysis information statistics. For an explanation of how to use the standard text display window, see **“Standard text display window” on page 37.**

Database statistics options

When you request a statistic option, InterBase generates and displays information for that database statistic. Possible statistic option values include: All Options, Data Pages, Database Log, Header Pages, Index Pages, and System Relations.

Note In addition to the selected statistic, header page information is displayed, regardless which statistic has been selected to report. If Header Pages is the selected option value, then only header page information will be displayed.

▸ *All Options*

Displays statistic information for all options including Data Pages, Database Log, Header Pages, Index Pages, and System Relations.

This function corresponds to the **-all** option of **gstat**.

Data Pages

Displays data page information in the database summary. Below is an example of data page information, followed by an explanation of each item.

```
COUNTRY (31)
   Primary pointer page: 246, Index root page: 247
   Data pages: 1, data page slots: 1, average fill: 59%
   Fill distribution:
   0 - 19% = 0
   20 - 39% = 0
   40 - 59% = 1
   60 - 79% = 0
   80 - 99% = 0
```

The first line displays a database table name while the remaining lines contain item information pertaining to the table. These items include:

Item	Description
Primary pointer page	The page that is the first pointer page for the table.
Index root page	The page number that is the first pointer page for indexes.
Data pages	The total number of data pages.
Data page slots	The number of pointers to database pages, whether the pages are still in the database or not.
Average fill	The average percentage to which the data pages are filled.
Fill distribution	A histogram that shows the number of data pages that are filled to the percentages.

TABLE 8.1 Data page information

Database Log

Displays the database log in the database summary. Below is an example of database log information.

This function corresponds to the **-log** option of **gstat**.

```
Database log page information:
   Creation date Dec 20, 1998 11:38:19
   Log flags:2
      No write ahead log
   Next log page:0
   Variable log data:
   Control Point 1:
      File name:
      Partition offset: 0 Seqno: 0 Offset: 0
   Control Point 2:
      File name:
      Partition offset: 0 Seqno: 0 Offset: 0
   Current File:
      File name:
      Partition offset: 0 Seqno: 0 Offset: 0
```

Header Pages

Displays header page information in the database summary. Below is an example of database summary header page information, followed by an explanation of each item.

This function corresponds to the **-header** option of **gstat**.

```
Database 'C:\Program Files\InterBase Corp\
   InterBase\examples\employee.gdb'
Database header page information:
   Flags 0
   Checksum 15351
   Generation 174
   Page size 1024
   ODS version 9.0
   Oldest transaction 22
   Oldest active 166
   Oldest snapshot 166
   Next transaction 170
   Bumped transaction 1
   Sequence number 0
   Next attachment ID 0
   Implementation ID 2
   Shadow count 0
   Page buffers 0
   Next header page 0
   Database dialect 3
   Creation date Nov 10, 1999 16:24:31
   Attributes force write

Variable header data:
```

The first line displays the name and location of the primary database file while the remaining lines contain information on the database header page. These items include:

Item	Description
Checksum	The header page checksum. This is a unique value computed from all the data in the header page. When the header page is stored to disk and later retrieved, the checksum of the retrieved page is recomputed and compared to the stored value to ensure that the information is correct.
Generation	Counter incremented each time header page is written.
Page size	The current database page size, in bytes.
ODS version	The version of the database's on-disk structure.
Oldest transaction	The transaction ID number of the oldest "interesting" transaction (those that are not committed, but active, in limbo, or rolled back).
Oldest active	The transaction ID number of the oldest active transaction.
Next transaction	The transaction ID number that InterBase assigns to the next transaction. The difference between the oldest transaction and the next transaction determines when database sweeping occurs. For example, if the difference is greater than this difference (set to 20,000 by default), then InterBase initiates a database sweep. See "Overview of sweeping" on page 121.
Sequence number	The sequence number of the header page (zero is used for the first page, one for second page, and so on).
Next connection ID	ID number of the next database connection.

TABLE 8.2 Header page information

Item	Description
Implementation ID:	The architecture of the system on which the database was created. These ID definitions are platform-dependent #define directives for a macro class named CLASS: • 1 HP Apollo Domain OS • 2 Sun Solaris SPARC, HP9000 s300, Xenix, Motorola IMP UNIX, UnixWare, NCR UNIX, NeXT, Data General DG-UX Intel • 3 Sun Solaris x86 • 4 VMS • 5 VAX Ultrix • 6 MIPS Ultrix • 7 HP9000 s700/s800 • 8 Novell NetWare • 9 Apple Macintosh 680x0 • 10 IBM AIX POWER series, IBM AIX PowerPC • 11 Data General DG-UX 88K • 12 HP MPE/xl • 13 SGI IRIX • 14 Cray • 15 SF/1 • 16 Microsoft Windows 32-bit Intel • 17 IBM OS/2 • 18 Microsoft Windows 16-bit • 19 Linux Intel • 20 Linux SPARC
Shadow count	The number of shadow files defined for the database.
Number of cache buffers	The number of page buffers in the database cache.

TABLE 8.2 Header page information

Item	Description
Next header page	The ID of the next header page.
Database dialect	The SQL dialect of the database
Creation date	The date when the database was created.
Attributes	• force write—indicates that forced database writes are enabled. • no_reserve—indicates that space is not reserved on each page for old generations of data. This enables data to be packed more closely on each page and therefore makes the database occupy less disk space. • shutdown—indicates database is shut down.
Variable header data	• sweep interval • secondary file information

TABLE 8.2 Header page information

Index Pages

Displays index information in the database summary. Below is an example of index page information, followed by an explanation of each item.

```
Index CUSTNAMEX (2)
   Depth: 2, leaf buckets: 2, nodes: 27
   Average data length: 45.00, total dup: 0, max dup: 0
   Fill distribution:
        0 - 19% = 0
       20 - 39% = 0
       40 - 59% = 1
```

```
        60 - 79% = 0
        80 - 99% = 1
```

Item	Description
Index	The name of the index.
Depth	The number of levels in the index page tree. If the depth of the index page tree is greater than three, then sorting may not be as efficient as possible. To reduce the depth of the index page tree, increase the page size. If increasing the page size does not reduce the depth, then return it to its previous size.
Leaf buckets	The number of leaf (bottom level) pages in the index page tree.
Nodes	The total number of index pages in the tree.
Average data length	The average length of each key, in bytes.
Total dup	The total number of rows that have duplicate indexes.
Max dup	The number of duplicates of the index with the most duplicates
Fill distribution	A histogram that shows the number of index pages filled to the specified percentages.

TABLE 8.3 Index pages information

System Relations

Displays information for system tables in the database.

```
RDB$CHECK_CONSTRAINTS (24)
    Primary pointer page: 54, Index root page: 55
    Data pages: 5, data page slots: 5, average fill: 59%
    Fill distribution:
        0 - 19% = 0
       20 - 39% = 1
       40 - 59% = 0
       60 - 79% = 4
       80 - 99% = 0

   Index RDB$INDEX_14 (0)
   Depth: 1, leaf buckets: 1, nodes: 68
   Average data length: 0.00, total dup: 14, max dup: 1
   Fill distribution:
```

```
 0 - 19% = 0
20 - 39% = 0
40 - 59% = 1
60 - 79% = 0
80 - 99% = 0
```

The statistics contained here are similar to that of data pages and index pages. For information on the items see **"Data Pages"** and **"Index Pages"** above.

gstat command-line tool

Syntax `gstat [options] database`

Description The **gstat** program is a command-line tool for retrieving and reporting database statistics. Its function is the same as that described for IBConsole earlier in this chapter.

You must be SYSDBA or the owner of a database to run **gstat**. On UNIX platforms, there is a further constraint on **gstat**: In order to run **gstat**, you must have system-level read access to the database files. You can gain this by logging in as the same account that the InterBase server is running as (interbase or root) or by setting the system-level permissions on the database file to include read permission for your Group. These restrictions exist on UNIX platforms because **gstat** accesses the database file at the system level rather than through the InterBase server.

Note You can run **gstat** only against local databases: run **gstat** on the server host.

Options TABLE 8.4 LISTS THE VALID OPTIONS FOR GSTAT.

Option	Description
-all	Equivalent to supplying **-index** and **-data**; this is the default if you supply none of **-index**, **-data**, or **-all**
-data	Retrieves and displays statistics on data tables in the database
-header	Stops reporting statistics after reporting the information on the header page
-index	Retrieves and displays statistics on indexes in the database

TABLE 8.4 **gstat** options

Option	Description
-log	Stops reporting statistics after reporting the information on the log pages
-pa[**ssword**] *text*	Checks for password *text* before accessing a database
-system	Retrieves statistics on system tables and indexes in addition to user tables and indexes
-user *name*	Checks for user *name* before accessing database
-z	Prints product version of **gstat**

TABLE 8.4 **gstat** options

Viewing lock statistics

Locking is a mechanism that InterBase uses to maintain the consistency of the database when it is accessed by multiple users. The lock manager is a thread in the **ibserver** process that coordinates locking.

The lock manager uses a lock table to coordinate resource sharing among client threads in the **ibserver** process connected to the database. The lock table contains information on all the locks in the system and their states. The global header information contains useful aggregate information such as the size of the lock table, the number of free locks, the number of deadlocks, and so on. There is also process information such as whether the lock has been granted or is waiting. This information is useful when trying to correct deadlock situations.

Syntax `iblockpr [a,o,w]` (Windows) or `gds_lock_print [a,o,w]` (UNIX)

`iblockpr [-i{a,o,w}] [`*`t n`*`]`

Description **iblockpr** monitors performance by checking lock requests.

The first form of syntax given above is to retrieve a report of lock statistics at one instant in time. The second form is to monitor performance by collecting samples at fixed intervals.

These options display interactive information on current activity in the lock table. t specifies the time in seconds between samplings. *n* (count) specifies the number of samples to be taken. The utility prints out the events per second for each sampling and gives the average of the values in each column at the end.

Options

Option	Description
[none]	Same as **-o**
-a	Prints a static view of the contents of the lock table
-o	Prints a static lock table summary and a list of all entities that own blocks
-w	Prints out all the information provided by the **-o** flag plus wait statistics for each owner; this option helps to discover which owner’s request is blocking others in the lock table
	The following options supply interactive statistics (events/second) for the requested items, which are sampled *n* times every *t* seconds, with one line printed for each sample. The average of the sample values is printed at the end of each column. If you do not supply values for *n* and *t*, the default is *n*=1.
-i	Prints all statistics; the output is easier to read if you issue **-ia**, **-io**, and **-iw** separatly
-ia	Prints how many threads are trying to acquire access to the lock table per second
-io	Prints operation statistics such lock requests, conversions, downgrades, and releases per second
-iw	Prints number of lock acquisitions and requests waiting per second, wait percent, and retries

TABLE 8.5 **iblockpr/gds_lock_print** options

Example The following statement prints “acquire” statistics (access to lock table: acquire/s, acqwait/s, %acqwait, acqrtry/s, and rtrysuc/s) every 3 seconds until 10 samples have been taken:

```
gds_lock_print -ia 3 10
```

Retrieving statistics programmatically

InterBase includes programming facilities to gather performance timings and database operation statistics.

You can use the API function *isc_database_info( )* to retrieve statistics, by specifying one or more of the following request buffer items:

Request Buffer Item	**Result Buffer Contents**
isc_info_fetches	Number of reads from the memory buffer cache; calculated since the InterBase server started
isc_info_marks	Number of writes to the memory buffer cache; calculated since the InterBase server started
isc_info_reads	Number of page reads; calculated since the InterBase server started
isc_info_writes	Number of page writes; calculated since the InterBase server started
isc_info_backout_count	Number of removals of record versions per table since the current database attachment started
isc_info_delete_count	Number of row deletions • Reported per table • Calculated since the current database attachment started
isc_info_expunge_count	Number of removals of a record and all of its ancestors, for records whose deletions have been committed • Reported per table • Calculated since the current database attachment started

TABLE 8.6 Database I/O statistics information items

Request Buffer Item	Result Buffer Contents
isc_info_insert_count	Number of inserts into the database • Reported per table • Calculated since the current database attachment started
isc_info_purge_count	Number of removals of old versions of fully mature records (records committed, resulting in older-ancestor-versions no longer being needed) • Reported per table • Calculated since the current database attachment started
isc_info_read_idx_count	Number of reads done via an index • Reported per table • Calculated since the current database attachment started
isc_info_read_seq_count	Number of sequential database reads, that is, the number of sequential table scans (row reads) • Reported per table • Calculated since the current database attachment started
isc_info_read_update_count	Number of row updates • Reported per table • Calculated since the current database attachment started

TABLE 8.6 Database I/O statistics information items

See the *API Guide* for information on request buffers, and details of using this API call.

CHAPTER

9

Interactive Query

This chapter documents the IBConsole interactive SQL (ISQL) and command-line **isql** utilities for InterBase. These tools provide an interface to InterBase's Dynamic SQL interpreter. You can use these query tools to perform data definition, prototype queries before implementing them in your application, or to perform ad hoc examination of data in your database.

Topics covered in this chapter include:

- **IBConsole ISQL**
- **Executing SQL statements**
- **Committing and rolling back transactions**
- **Saving ISQL input and output**
- **Changing ISQL settings**
- **Extracting metadata**
- **Command-line isql tool**

IBConsole ISQL

The IBConsole ISQL Window permits you to execute DDL and DML commands to the InterBase server as well as to load, save, print, cut, paste, and copy SQL scripts and results.

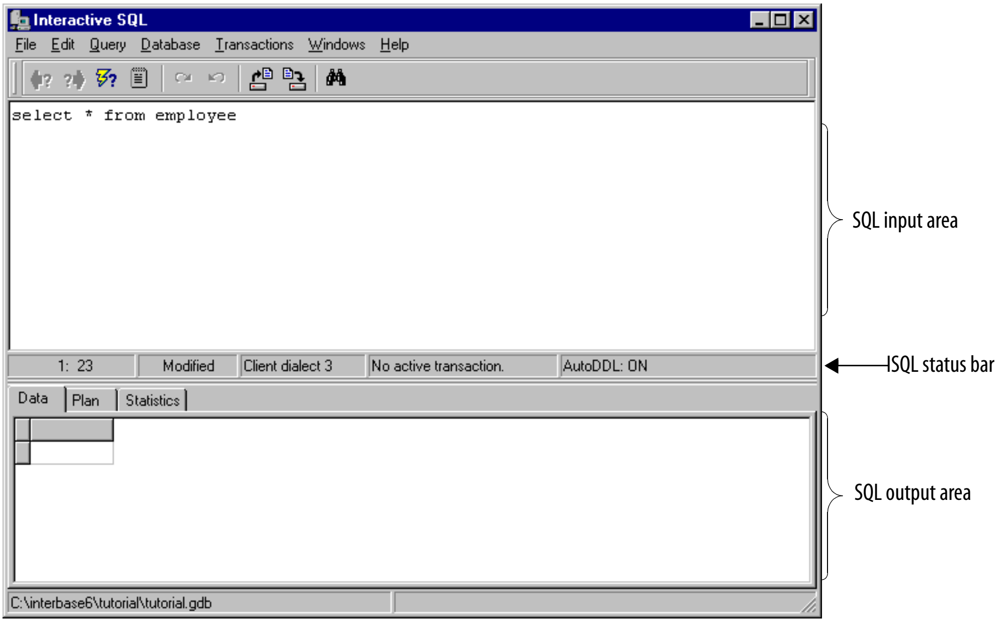
The ISQL window

To access the ISQL Window:

- Click the Launch SQL toolbar button
- Choose **Tools | Interactive SQL.**

The ISQL window appears.

FIGURE 9.1 IBConsole - ISQL



```
select * from employee
```

1: 23 | Modified | Client dialect 3 | No active transaction. | AutoDDL: ON

C:\interbase6\tutorial\tutorial.gdb

ISQL menus

Menus that are unique to IBConsole ISQL are the Edit, Query, and Transaction menus.

EDIT MENU

Edit menu items include undo, cut, copy, paste, find, and select all. You can use all Edit menu items while working in the SQL input area. You can use all Edit menu items while working in the SQL output area except Undo, Cut, and Paste. "Undo" in the Edit menu does not undo database changes. Use **Transactions | Rollback** to abort database changes. The Edit menu also includes Options, which allows you to change ISQL session settings. For more information on the Options dialog, see **"Using InterBase Manager to start and stop InterBase" on page 43**.

QUERY MENU

Query menu items enable you to perform ISQL commands. These menu items include Load Script, Save Script, Next, Previous, Execute, Prepare and Save Output.

TRANSACTIONS MENU

Transactions menu items enable you to commit and rollback database changes.

▸ *ISQL toolbar*

See **TABLE 9.1, "Toolbar buttons for executing SQL statements," on page 191** for a description of each toolbar button included in ISQL toolbar.

▸ *ISQL work areas*

SQL INPUT AREA

The SQL input area is where you can type SQL statements or scripts to be executed. It scrolls vertically.

SQL OUTPUT AREA

The SQL output area is where the results of the SQL statements or scripts are displayed. It scrolls both horizontally and vertically. The SQL output area contains two tabs:

- The Data tab displays any data returned by the SQL output in a grid format.
- The Plan tab displays the plan for the SQL statement or script.
- The Statistics tab displays statistics for the SQL output, including: execution time, prepare time, starting memory, delta memory, current memory, number of buffers, number of reads, number of writes, the plan, and the number of records fetched.

▸ *Status bar*

The status bar at the bottom of the SQL input area displays information relevant to the SQL input areas such as cursor position, input status, client dialect, and transaction status. You can change the client dialect by right clicking on the status bar.

Managing ISQL temporary files

ISQL creates temporary files used during a session to store information such as the command history, output file names, and so on. These files are named in the form **isql_aa.xx**. The files are stored in the directory specified by the TMP environment variable, or if that is not defined, the working directory, or if that is not defined, they are stored in the **Windows** directory.

To avoid cluttering the **Windows** directory with InterBase temporary files, specify a different directory for them by defining TMP.

When you exit, ISQL deletes these temporary files. If ISQL abnormally terminates (for example, due to a power failure), then these files remain and may be freely deleted without any adverse effects. You should not delete any of these temporary files while ISQL is running, because they may be used in the current session.

Executing SQL statements

In ISQL, you can execute SQL statements in either of two ways:

- Interactively, one statement at a time
- From a script containing multiple statements

Executing SQL interactively

To execute an SQL statement interactively:

1. Type a single SQL statement in the SQL Input area. Make sure any other existing statements are *commented*. A statement is commented if it is preceded by "/*" and followed by "*/".

   If the statement already exists in the SQL Input area make sure all statements except the one you wish to execute are commented. Commented statements in the SQL Input area are ignored during execution.

2. Choose **Query | Execute**, enter Ctrl+E, or click the Execute toolbar button.

If more than one statement is uncommented, Execute executes each statement, one after the other.

Tip You can copy text from other Windows applications such as the Notepad and Wordpad text editors and paste it into the SQL Input area. You can also copy statements from the ISQL Output area and paste them into the SQL Input area. This cut-and-paste method is also a convenient way to use the online SQL tutorial provided in the online Help.

When SQL statements are executed, whether successfully or not, they become part of the ISQL *command history*, a sequential list of SQL statements entered in the current session.

The buttons in the Toolbar pertaining to execution of SQL statements are:

Button	Menu Option	Description
⚡?	**Query | Execute**	Executes the current statement or script in the SQL input area. The resultant output is displayed in the SQL Output area. The accelerator key is Ctrl-E.
⇦?	**Query | Previous**	Recalls the previous SQL statement in the command history. If no previous statement exists in the command history, this button is disabled. The accelerator key is Ctrl-P.
?⇨	**Query | Next**	Recalls the next SQL statement in the command history. If no next statement exists in the command history, this button is disabled. The accelerator key is Ctrl-N.
📋	**Query | Prepare**	Prepares the query for execution. The query plan is displayed in the SQL output area.
↷	**Transactions | Commit**	Commits the transaction specified by the SQL statement to the database.
↶	**Transactions | Rollback**	Rolls back all database changes since the last commit.
↱📄	**Query | Load Script**	Loads a script for SQL execution into the SQL input area.
📄↴	**Query | Save Script**	Saves SQL statements entered in the SQL input area to a file.

TABLE 9.1 Toolbar buttons for executing SQL statements

Preparing SQL statements

Use the Prepare toolbar button, or select **Query | Prepare**, to prepare SQL statements for execution and to view the query plan. Prepare compiles the query plan on the server, and displays it in the Plan tab of the SQL output area. Use Prepare to determine if your SQL script is well-constructed, without having to wait for the SQL script to execute.

▶ *Valid SQL statements*

- You can execute interactively any SQL statement identified as "available in DSQL" in the *Language Reference*. You cannot use any statements that are specifically identified in the *Language Reference* as **isql** statements; all these have functionally equivalent menu items in ISQL.

  For example, the SET NAMES statement cannot be executed from the SQL Input area. To change the active character set, choose **Edit | Options** and select the desired character set option value in the SQL Options dialog.
- SQL script files can include statements that are not valid to enter interactively. For example, you can use the SET statements such as SET LIST or SET TERM in scripts.
- Transaction names may not be used with SET TRANSACTION statement.
- The SQL Input area accepts multiple statements, although only one can be executed at a time. Each statement entered in the SQL input area must be terminated by a semicolon (**;**). The SQL Input area accepts multiple statements, although only one can be executed at a time. An uncommented statement that holds the mouse cursor is called the *current statement*.

▶ *Loading and executing an SQL script file*

To execute an SQL script file containing SQL statements:

1. Choose **Query | Load Script** or click the Load Script toolbar button.
2. Locate the desired script file in the Open dialog, and click Open to display the statements of the script file in the SQL Input area.
3. Ensure that you are connected to the desired database.
4. If you are connected to the database, comment out any CONNECT or CREATE DATABASE statements.
5. Choose **Query | Execute** or click **Execute** on the toolbar to begin executing the entire script statement by statement.

Note Statements executed from a loaded script file do not become part of the command history.

Committing and rolling back transactions

Changes to the database from data definition (DDL) statements—for example, CREATE and ALTER statements—are automatically committed by default. To turn off automatic commit of DDL, choose **Edit | Options** and set the Auto Commit DDL option to false in the SQL Options dialog.

Changes made to the database by data manipulation (DML) statements—for example INSERT and UPDATE—are not permanent until they are committed. Commit changes by choosing **Transactions | Commit** or by clicking Commit on the toolbar.

To undo all database changes from DML statements since the last commit, choose **Transactions | Rollback** or click Rollback on the toolbar.

Saving ISQL input and output

You can save the following to a file:

- SQL statements entered in the SQL Input area of the current session.
- The output of the last SQL statement executed.

▸ *Saving SQL input*

To save the SQL statements entered in the SQL Input area of the current session to a text file:

1. Choose **Query | Save Script** or click the Save Script toolbar button.
2. Enter a file name, including the location for the new file, in the Save As dialog and click Save.

   To include the location for the file, type the file path and file name in the Filename text area, or browse to the folder where you would like the file to reside and type only the file name.

Only the SQL statements entered in the current session, not the output, are saved to the specified file.

▸ *Saving SQL output*

To save the results of the last executed SQL statement to a file:

1. Choose **Query | Save Output to File**.
2. Enter a file name, including the location for the new file, in the Export To dialog and click Save.

To include the location for the file, either type the file path and file name in the Filename text area, or browse to the folder where you would like the file to reside and type only the file name.

The output in the Data tab from the last successful statement is saved to the named text file.

If you run an SQL script, and then choose to save the output, all the commands in the script file and their results are saved to the output file. If command display has been turned off in a script with SET ECHO OFF, then SQL statements in the script are not saved to the file.

Changing ISQL settings

Use the SQL Options dialog to display and modify ISQL session settings, determine if the main IBConsole window will be updated based on the statements given in the ISQL window, and specify how open transactions are handled when the ISQL window is closed.

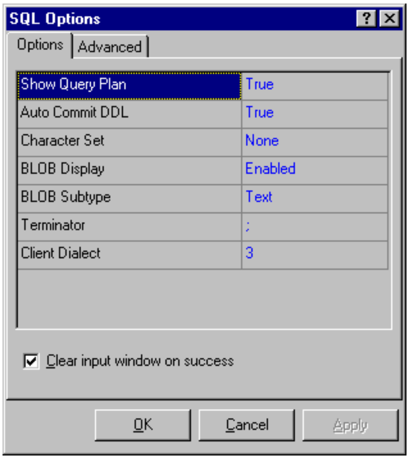
Select **Edit | Options** from the Interactive SQL window to display the SQL Options dialog.

The SQL Options dialog has two tabs: Options and Advanced.

Options tab

Use the Options tab to display and modify the ISQL session settings. You can specify options by clicking the option value and choosing a new value from a drop down list of values or by double-clicking the option value to rotate its value to the next in the list of values.

FIGURE 9.2 Options tab of the SQL Options dialog



Show Query Plan	True
Auto Commit DDL	True
Character Set	None
BLOB Display	Enabled
BLOB Subtype	Text
Terminator	;
Client Dialect	3

☑ Clear input window on success

The following table summarizes the **isql** session settings available on the Options tab.

Settings and Values	**Behavior**
Show Query Plan: true (default) or false	f this setting is True, IBConsole displays the query plan chosen by the optimizer when a SELECT statement is entered. To modify the optimizer plan, use the PLAN option of the SQL SELECT statement. See **“SET PLAN”** on page 228.
Auto Commit DDL • True (default) • False	• If this setting is True, IBConsole automatically commits DDL (data definition) statements as each statement is entered. • If this setting is False, you must explicitly commit DDL statements (with **Transactions | Commit**) to make them permanent. See **“SET AUTODDL”**on page 221.

TABLE 9.2 Options Tab of the SQL Options dialog

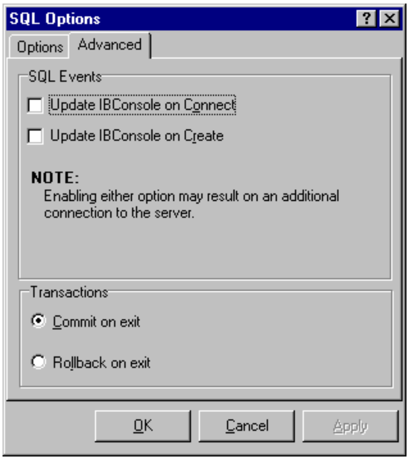
Settings and Values	Behavior
Terminator	Identifies the end-of-statement symbol to be used for SQL queries • Default terminator is the semicolon (;) • You can specify any character or group of characters as the terminator. You can change the terminator with the -**terminator** command line option or with the SET TERMINATOR command in an SQL script.
Client Dialect: • 1, 2, or 3(default)	Sets the IBConsole client dialect. See "Understanding SQL dialects" in *Getting Started* for more information
Character Set	Determines the active character set for strings for subsequent connections to the database; enables you to override the default character set for a database. • Specify the character set before connecting to the database whose character set you want to specify. For a complete list of character sets recognized by InterBase, see the *Language Reference*. • Your choice of character set limits possible collation orders to a subset of all available collation orders. Given a character set, a collation order can be specified when data is selected, inserted, or updated in a column. • You can perform the same function in an SQL script with the SET NAMES command. Use SET NAMES before connecting to the database whose character set you want to specify. See **"SET NAMES"** on page 227 for more information.
BLOB Display • Enabled (default) • Disabled Restrict	Determines how IBConsole displays columns of Blob data. SELECT always displays the Blob ID for columns of Blob datatype. By default, a SELECT also displays actual Blob data of text subtypes beneath the associated row. • If this setting is set to Enabled, IBConsole displays the contents of Blob columns. • If this setting is set to Disabled, IBConsole does not display the contents of Blob columns. If this setting is set to Restrict, IBConsole displays the contents of only Blob columns of the specified BLOB Subtype.
Clear input window on success	Check this box to clear the SQL input window after an SQL statement is successfully executed

TABLE 9.2 Options Tab of the SQL Options dialog

Advanced tab

Use the Advanced tab to determine if the main IBConsole window will be updated based on the statements given in the ISQL window, and to specify how open transactions are handled when the ISQL window is closed.

FIGURE 9.3 Advanced tab of the SQL Options dialog



SQL Events

□ Update IBConsole on Connect

□ Update IBConsole on Create

NOTE:
Enabling either option may result on an additional connection to the server.

Transactions

● Commit on exit

○ Rollback on exit

The following table summarizes the settings available on the Advanced tab:

Setting and values	Behavior
SQL Events	• Update IBConsole on Connect does not update the connected databases in the IBConsole window. However, it ensures that that the statements executed in the ISQL window are reflected in the information in the IBConsole window. • Update IBConsole on Create updates the main window if the currently selected server is active. Any database created in the ISQL window will automatically registered and an alias will be created.
Transactions • Commit on exit • Rollback on exit	Either commits or rolls back any active transactions, depending on the selection, when the ISQL window is closed.

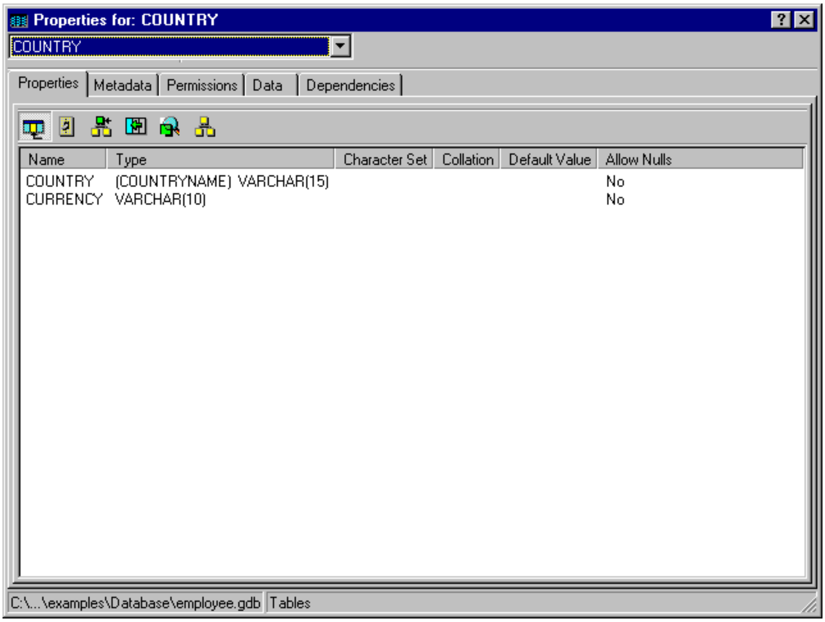
TABLE 9.3 Advanced tab of the SQL Options dialog

Inspecting database objects

Use the object inspector to view properties, metadata, permissions, data, and dependencies for the entire database or for a specific table, view, function, procedure, or any other database attribute displayed in the Tree pane.

To open the object inspector, double-click a database object in the Work pane. The object inspector appears:

FIGURE 9.4 Object inspector



Name	Type	Character Set	Collation	Default Value	Allow Nulls
COUNTRY	(COUNTRYNAME) VARCHAR(15)				No
CURRENCY	VARCHAR(10)				No

C:\...\examples\Database\employee.gdb Tables

Depending on the database object selected, the object inspector has some or all of the following tabs: Properties, Metadata, Permissions, Data, and Dependencies. These are discussed in the following sections.

Viewing object properties

The Properties tab is available when viewing Table and View database objects. Use the Properties tab of the object inspector to display properties for database objects, including columns, triggers, check constraints, indexes, unique constraints, and referential constraints. The Properties tab has five toolbar buttons for displaying the various object properties:

Button	Description
	Show columns: displays the name, type, collation, character set, default value, and whether or not null values are acceptable for every row in the column. The accelerator key is Ctrl+Alt+C. For more information on columns, refer to “Defining columns” in the *Data Definition Guide*.
	Show triggers: displays the name and type of each trigger, as well as whether or not it is active. In addition, it displays the SQL trigger statement. The accelerator key is Ctrl+Alt+T. For more information on triggers, refer to Chapter 10: "Working with Triggers" in the *Data Definition Guide*.
	Show check constraints: displays the names of the constraints, whether or not they can be deferred, and if they were initially deferred. In addition, it displays the SQL check constraint statements. The accelerator key is Ctrl+Alt+H. For more information, refer to “Defining a CHECK constraint” in the *Data Definition Guide*.
	Show indexes: displays the name of the index keys, and whether or not they are unique, descending, or active. The accelerator key is Ctrl+Alt+R. For more information, refer to Chapter 7: "Working with Indexes" in the *Data Definition Guide*.
	Show unique constraints: displays the names of the constraints, whether or not they can be deferred, if they were initially deferred, and the index keys. The accelerator key is Ctrl+Alt+U.
	Show referential constraints: displays the names of the constraints, whether or not they can be deferred, if they were initially deferred, the update rule, the update rules, the delete rules, the index, and the reference table. The accelerator key is Ctrl+Alt+R.

TABLE 9.4 Object inspector toolbar buttons

Viewing metadata

The metadata which the Metadata tab of the object inspector displays depends on the database that is selected in the Tree pane, or the item that is selected in the Work pane.

To view metadata for an entire database Select a connected database in the Tree pane, and then click on View Metadata in the Work pane. The metadata is displayed in a text window.

To view metadata for a specific database object Select a database element from the hierarchy displayed in the Tree pane, and then right-click a database object associated with that element in the Work pane and select Extract from the context menu.

You can also double-click a database object in the Work pane to view metadata.

If you want metadata for domains only, expand the desired database hierarchy (if it is not already expanded), select Domains, click on a domain element in the Work pane, and select the Metadata tab of the object inspector.

If you want metadata for roles only, expand the desired database hierarchy (if it is not already expanded), select Roles and click on a role element in the Work pane, and select the Metadata tab of the object inspector.

Use the drop down list at the top of the dialog to select other objects associated with the database element.

The following table lists the items for which you can view metadata for associated objects with the object inspector.

Item	Displays
Blob Filters	Blob filters definition
Domains	Metadata script, dependencies, datatype, description, check constraints, and default values
Exceptions	Description, exception number, exception message, metadata script, and dependencies
External Functions	UDFs definition
Generators	Generator ID, current value, metadata script, and dependencies
Stored Procedures	Metadata script, procedure body, input parameters, output parameters, permissions, data, and dependencies
Roles	Role definition
Tables	Columns, datatypes, triggers, indexes, unique constraints, referential constraints, check constraints, metadata script, permissions, data, and dependencies
Views	Metadata script, permissions, data, and dependencies

TABLE 9.5 Metadata information items

▸ *Extracting metadata*

You can extract a metadata script to a file by displaying the desired metadata in the Metadata tab and clicking the Save Script toolbar button.

Extracting an entire database exports metadata in a specific order, to allow the resulting script to be used as input to recreate the database.

Metadata	Comments
1. Database	Extracts database with default character set and PAGE_SIZE
2. Domains	Must be before tables that reference domains
3. Tables	Must be after domains
4. Indexes	Must be after tables
5. FOREIGN KEY constraints	Must be added after tables to avoid tables being referenced before they have been created
6. Views	Must be after tables
7. CHECK constraints	Must be after tables
8. Exceptions	Must be extracted before stored procedures and triggers that contain code to raise exceptions
9. Stored procedures	Stored procedures are shown with no body in CREATE PROCEDURE and then ALTER PROCEDURE to add the text of the procedure body; this is to allow circular or recursive procedure references
10. Triggers	Must be after tables Must be after stored procedures, to allow trigger code to reference procedures Does not extract triggers from CHECK constraints
11. Roles	Must be before GRANT privileges
12. GRANTS	Must be after tables, views, stored procedures, triggers, and roles

TABLE 9.6 Metadata extraction constraints

Items that are not extracted include:

- Code of external functions or filters, because that code is not part of the database. The declarations to the database (with DECLARE EXTERNAL FUNCTION and DECLARE FILTER) are extracted
- System tables, system views, and system triggers

- Because DDL statements do not contain references to object ownership, the extracted file does not show ownership. The output file includes the name of the object and the owner if one is defined. There is no way to assign an object to its original owner.

Extracting metadata

You can extract a metadata script to a file by displaying the desired metadata in the Metadata tab and clicking the Save Script toolbar button.

Extracting an entire database exports metadata in a specific order, to allow the resulting script to be used as input to recreate the database.

Metadata	Comments
1. Database	Extracts database with default character set and PAGE_SIZE
2. Domains	Must be before tables that reference domains
3. Tables	Must be after domains
4. Indexes	Must be after tables
5. FOREIGN KEY constraints	Must be added after tables to avoid tables being referenced before they have been created
6. Views	Must be after tables
7. CHECK constraints	Must be after tables
8. Exceptions	Must be extracted before stored procedures and triggers that contain code to raise exceptions
9. Stored procedures	Stored procedures are shown with no body in CREATE PROCEDURE and then ALTER PROCEDURE to add the text of the procedure body; this is to allow circular or recursive procedure references
10. Triggers	Must be after tables Must be after stored procedures, to allow trigger code to reference procedures Does not extract triggers from CHECK constraints.
11. Roles	Must be before GRANT privileges
12. GRANTs	Must be after tables, views, stored procedures, triggers, and roles

TABLE 9.7 Order of metadata extraction

Items that are not extracted include:

- Code of external functions or filters, because that code is not part of the database. The declarations to the database (with DECLARE EXTERNAL FUNCTION and DECLARE FILTER) are extracted.
- System tables, system views, and system triggers.
- Because DDL statements do not contain references to object ownership, the extracted file does not show ownership. The output file includes the name of the object and the owner if one is defined. There is no way to assign an object to its original owner.

Command-line isql tool

Command-line **isql** is a utility for processing SQL data definition (DDL) and data manipulation (DML) statements from interactive input or from a source file. It enables you to create and view metadata, add and modify data, grant user permissions, test queries, and perform database administration tasks.

This section provides an introduction to using **isql**. For a description of the standard SQL commands available in **isql**, see the *Language Reference*. For a description of special **isql** commands, see **"isql command reference" on page 213**.

Invoking isql

You can use **isql** in the following ways:

- Interactively to process SQL statements, by entering statements at the **isql** prompt
- Noninteractively to process SQL statements in a file

To start the isql utility, type the following at a UNIX shell prompt or Windows console prompt:

```
isql [options] [database_name]
```

where **options** are command-line options and *database_name* is the name of the database to connect to, including disk and directory path.

If no options are specified, **isql** starts an interactive session. If no database is specified, you must connect to an existing database or create a new one. If a database was specified, **isql** starts the interactive session by connecting to the named database.

If options are specified, **isql** starts interactively or noninteractively, depending on the options. For example, reading an input file and writing to an output file are noninteractive tasks, so the **-input** or **-output** options do not start an interactive session. Additional noninteractive options include **-a**, **-database**, **-extract**, and **-x**, which are used when extracting DDL statements.

When you start an interactive **isql** session, the following prompt appears:

```
SQL>
```

You must then end each command with a terminator character. The default terminator is a semicolon (**;**). You can change the terminator to any character or group of characters with the SET TERMINATOR command or with the **-terminator** command-line option. If you omit the terminator, a continuation prompt appears (`CON>`).

Note For clarity, all of the commands and examples in this chapter end with the default semicolon terminator.

▸ *Command-line options*

Only the initial characters in an option are required. You can also type any portion of the text enclosed in brackets, including the full option name. For example, specifying **-n**, **-no**, or **-noauto** has the same effect.

Option	Description
-a	Extracts all DDL for the named database
-d[**atabase**] *name*	Used with **-x**; changes the CREATE DATABASE statement that is extracted to a file • Without **-d**, CREATE DATABASE appears as a C-style comment and uses the database name specified on the **isql** command line • With **-d**, **isql** extracts an uncommented CREATE DATABASE and substitutes *name* as its database argument
-c[**ache**]	Set number of cache buffers for this connection to the database; see **"Default cache size per ISQL connection" on page 125**.
-e[**cho**]	Displays (echoes) each statement before executing it
-ex[**tract**]	Same as **-x**

TABLE 9.8 **isql** command-line options

Option	Description
-i[**nput**] *file*	Reads commands from an input file instead of from standard input • Input files can contain **-input** commands that call other files, enabling execution to branch and then return • **isql** exits (with a commit) when it reaches the end of the first file • In interactive sessions, use **-input** to read commands from a file
-m[**erge_stderr**]	• Merges stderr output with stdout • Useful for capturing output and errors to a single file when running **isql** in a shell script or batch file
-n[**oauto**]	Turns off automatic commitment of DDL statements; by default, DDL statements are committed automatically in a separate transaction
-nowarnings	Displays warning messages if and only if an error occurs (be default, **isql** displays any message returned in a status vector, even if no error occurred)
-o[**utput**] *file*	Writes results to an output file instead of to standard output; in interactive sessions, use **-output** to write results to a file
-pas[**sword**] *password*	Used with **-user** • Specifies a password when connecting to a remote server • For access, both *password* and *user* must represent a valid entry in the security database
-page[**length**] *n*	Prints column headers every *n* lines instead of the default 20
-q[**uiet**]	
-r[**ole**] *rolename*	Grants privileges of role *rolename* to *user* on connection to the database

TABLE 9.8 **isql** command-line options (*continued*)

Option	Description
-s[qldialect] *n*	Interprets subsequent commands as dialect *n* until end of session or until dialect is changed by a SET SQL DIALECT statement • For *n* = 1, commands are processed as in InterBase 5 • For *n* = 2, elements that have different interpretations in dialect 1 and 3 are all flagged with warnings or errors to assist in migrating databases to dialect 3 • For *n* = 3, all statements are parsed as InterBase 6 SQL semantics: double quotes are delimited identifiers, DATE datatype is SQL DATE, and exact numerics with precision greater than 9 are stored as INT64
-t[**erminator**] *x*	Changes the end-of-statement symbol from the default semicolon (;) to *x*, where *x* is a single character or any sequence of characters
-u[**ser**] *user*	Used with **-password**; specifies a user name when connecting to a remote server • For access, both *password* and *user* must represent a valid entry in the security database
-x	Extracts DDL for the named database; displays DDL to the screen unless redirected to a file
-z	Displays the software version of **isql**

TABLE 9.8 **isql** command-line options (*continued*)

Using warnings

Warnings can be issued for the following conditions:

- SQL statements with no effect
- SQL expressions that produce different results in InterBase 5 versus InterBase 6
- API calls that will be replaced in future versions of the product
- Pending database shutdown

Examples

- Suppose **createdb.sql** contains DDL statements to create a database. To execute the statements, enter:

```
isql -input createdb.sql
```

- The following example starts an interactive connection to a remote database. The remote server, jupiter, accepts the specified user and password combination with the privileges assigned to the STAFF role:

  ```
  isql -user sales -password mycode -role 'staff'
     'jupiter:/usr/customer.gdb'
  ```

- The next example starts an interactive session but does not attach to a database. **isql** commands are displayed, and query results print column headers every 30 lines:

  ```
  isql -echo -page 30
  ```

Exiting isql

To exit **isql** and roll back all uncommitted work, enter:

```
QUIT;
```

To exit **isql** and commit all work, enter:

```
EXIT;
```

Connecting to a database

If you do not specify a database on the command-line when invoking **isql**, you must either connect to an existing database or create a new one. Use the CONNECT command to connect to a database and CREATE DATABASE to create a database. For the full syntax of CONNECT and CREATE DATABASE, see the *Language Reference*.

You can connect to a database in two ways. You can connect to:

- A local database on a Windows platform. Use the CONNECT command with the full path of the database as the argument. For example:

  ```
  SQL> CONNECT 'C:/InterBase/examples/employee.gdb' role 'staff';
  ```

- A remote database on a Windows or UNIX server using TCP/IP. Use the CONNECT command with the full node name and path of the database as the argument. Separate the node name from the database path with a colon.

  To connect to a database on a UNIX platform named jupiter:

  ```
  SQL> CONNECT 'jupiter:/usr/interbase/examples/employee.gdb';
  ```

  To connect to a database on a Windows 2000 platform named venus:

  ```
  SQL> CONNECT 'venus:c:/InterBase/InterBase
     /examples/database/employee.gdb';
  ```

Note Be careful not to confuse node names and shared disks, since both are specified with a colon separator. If you specify a single letter that maps to a disk drive, it is assumed to be a drive, not a node name.

Tip You can use either forward slashes (/) or backslashes (\ ) as directory separators. InterBase automatically converts either type of slash to the appropriate type for the server operating system.

Setting isql client dialect

To use **isql** to create a database in a particular dialect, first set **isql** to the desired dialect and then create the database. You can set **isql** dialect the following ways:

- On the command line, start **isql** with option **-sql_dialect** *n*, where *n* is 1, 2, or 3:

```
isql -sql_dialect n
```

- Within an **isql** session or in an SQL script, include the following statement:

```
SET SQL DIALECT n;
```

isql dialect precedence is as follows:

Lowest: Dialect of an attached Version 6 database

Next lowest: Dialect specified on the command line

Next highest: Dialect specified during the session

Highest: Dialect of an attached Version 5 database (=1)

In InterBase 6, **isql** has the following behavior with respect to dialects:

- If you start **isql** and attach to a database withoutspecifying a dialect, **isql** takes on the dialect of the database.
- If you specify a dialect on the command line when you invoke **isql**, it retains that dialect after connection unless explicitly changed.
- When you change the dialect during a session using SET SQL DIALECT *n*, **isql** continues to operate in that dialect until it is explicitly changed.
- When you create a database using **isql**, the database is created with the dialect of the **isql** client; for example, if **isql** has been set to dialect 1, when you create a database, it is a dialect 1 database.
- If you create a database without first specifying a dialect for the **isql** client or attaching to a database, **isql** creates the database in dialect 3.

The statements above are true whether you are running **isql** as a command-line utility or accessing it through IBConsole.

IMPORTANT Any InterBase 6 **isql** client that attaches to a Version 5 database resets to dialect 1.

Transaction behavior in isql

When you start **isql**, InterBase begins a transaction. That transaction remains in effect until you issue a COMMIT or ROLLBACK statement. You must issue a COMMIT or ROLLBACK statement to end a transaction. Issuing one of these statements automatically starts a new transaction. You can also start a transaction with the SET TRANSACTION statement.

isql uses a separate transaction for DDL statements. When these statements are issued at the `SQL>` prompt, they are committed automatically as soon as they are completed. DDL scripts should issue a COMMIT after every CREATE statement to ensure that new database objects are available to all subsequent statements that depend on them. For more information on DDL statements, see the *Data Definition Guide*.

Extracting metadata

You can extract the DDL statements that define the metadata for a database to an output file with the **-extract** option. Adding the optional **-output** flag reroutes output to a named file. Use this syntax:

```
isql [[-extract | -x][-a] [[-output | -o] outputfile]] database;
```

The **-x** option is an abbreviation for **-extract**. The **-a** flag directs **isql** to extract all database objects. Note that the output file specification, *outputfile*, must follow the **-output** flag, while you can place the name of the database being extracted at the end of the command.

Option	Description
database	File specification of the database from which metadata is being extracted
outputfile	File specification of the text file to receive the extracted statements; if omitted, **isql** writes the information to the screen

TABLE 9.9 **isql** extracting metadata arguments

You can use the resulting text file to:

- Examine the current state of a database's system tables before you plan alterations to it, or when a database has changed significantly since its creation.
- Use your text editor to make changes to the database definition or create a new database source file.

The **-extract** option does not extract UDF code and Blob filters, because they are not part of the database. It *does* extract the declarations to the database (with DECLARE EXTERNAL FUNCTION and DECLARE FILTER).

The **-extract** option also does not extract system tables, system views, or system triggers.

Because DDL statements do not contain references to object ownership, the extracted file does not show ownership. The output file includes the name of the object and the owner if one is defined. There is no way to assign an object to its original owner.

For a list of the order of extraction of metadata objects, see **"Extracting metadata" on page 203**.For example, the following statement extracts the system catalogs from the database **employee.gdb** to a file called **employee.sql**:

```
isql -extract -output employee.sql employee.gdb;
```

The resulting output script is created with **-commit** following each set of commands, so that tables can be referenced in subsequent definitions. This command extracts all keywords and object names in uppercase when possible (some international metadata has no uppercase).

To extract DDL statements from database **employee.gdb** and store in the file **employee.sql**, enter:

```
isql -a employee.gdb -output employee.sql
```

The following example extracts the DDL statements from the database **dev.gdb**:

```
isql -x dev.gdb
```

This example combines the **-extract** and **-output** options to extract the DDL statements from the database **dev.gdb** into a file called **dev.out**. The output database name must follow the **-output** flag.

```
isql -extract -output dev.out dev.gdb
```

isql Commands

At the `SQL>` prompt, you can enter any of three kinds of commands:

- SQL data definition (DDL) statements, such as CREATE, ALTER, DROP, GRANT, and REVOKE. These statements create, modify, or remove metadata and objects, and control user access (via privileges) to the database. For more information about DDL, see the *Data Definition Guide*.

- SQL data manipulation (DML) statements such as SELECT, INSERT, UPDATE, and DELETE. These four data manipulation operations affect the data in a database. They retrieve, modify, add, or delete data. For more information about DML statements, see the *Language Reference*.
- **isql** commands that fall into three main categories:
  - SHOW commands (to display metadata or other database information)
  - SET commands (to modify the **isql** environment)
  - Other commands (for example, commands to read an input file, write to an output file, or end an **isql** session)

  Some **isql** commands have many options. **See "isql command reference" on page 213.**

SHOW *commands*

SHOW commands are used to display metadata, including tables, indexes, procedures, and triggers.

SHOW commands list all of the specified objects or give information about a particular object when used with *name.*

SHOW commands operate on a separate transaction from user statements. They run as READ COMMITTED background statements and acknowledge all metadata changes immediately.

SET *commands*

SET commands enable you to view and change the **isql** environment.

Other isql commands

The remaining **isql** commands perform a variety of useful tasks, including reading an SQL file, executing shell commands, and exiting **isql**. The other **isql** commands are: BLOBDUMP, EDIT, EXIT, HELP, INPUT, OUTPUT, QUIT, SHELL.

Exiting isql

To exit the **isql** utility and roll back all uncommitted work, enter:

```
SQL> QUIT;
```

To exit the **isql** utility and commit all work, enter:

```
SQL> EXIT;
```

Error handling

InterBase handles errors in **isql** and DSQL in the same way. To indicate the causes of an error, **isql** uses the SQLCODE variable and the InterBase status array.

The following table lists values that are returned to SQLCODE:

SQLCODE	Message	Meaning
< 0	SQLERROR	Error occurred; statement did not execute
0	SUCCESS	Successful execution
+1–99	SQLWARNING	System warning or informational message
+100	NOT FOUND	No qualifying rows found, or end of current active set of rows reached

TABLE 9.10 SQLCODE and message summary

For a detailed discussion of error handling, see the *Embedded SQL Guide*. For a complete listing of SQLCODE and InterBase status array codes, see the *Language Reference*.

isql command reference

This chapter describes the syntax and usage for commands available only in InterBase **isql** (interactive SQL). These commands are also available in SQL scripts. For a description of the standard DSQL commands available in **isql**, see the *Language Reference*.

Command-line **isql** supports the following special commands:

BLOBDUMP	SET BLOBDISPLAY	SHELL	SHOW INDEX
EDIT	SET COUNT	SHOW CHECK	SHOW INDICES
EXIT	SET ECHO	SHOW DATABASE	SHOW PROCEDURES
HELP	SET LIST	SHOW DOMAINS	SHOW ROLES

TABLE 9.11 **isql** commands

INPUT	SET NAMES	SHOW EXCEPTIONS	SHOW SYSTEM
OUTPUT	SET PLAN	SHOW FILTERS	SHOW TABLES
QUIT	SET STATS	SHOW FUNCTIONS	SHOW TRIGGERS
SET	SET TERM	SHOW GENERATORS	SHOW VERSION
SET AUTODDL	SET TIME	SHOW GRANT	SHOW VIEWS

TABLE 9.11 **isql** commands (*continued*)

BLOBDUMP

Places the contents of a BLOB column in a named file for reading or editing.

Syntax

```
BLOBDUMP blob_id filename;
```

Argument	Description
blob_id	System-assigned hexadecimal identifier, made up of two hexadecimal numbers separated by a colon (:) • First number is the ID of the table containing the BLOB column • Second number is a sequential number identifying a particular instance of Blob data
filename	Name of the file into which to place Blob contents

Description BLOBDUMP stores Blob data identified by *blob_id* in the file specified by *filename*. Because binary files cannot be displayed, BLOBDUMP is useful for viewing or editing binary data. BLOBDUMP is also useful for saving blocks of text (Blob data) to a file.

To determine the blob_id to supply in the BLOBDUMP statement, issue any SELECT statement that selects a column of Blob data. When the table's columns appear, any Blob columns contain hexadecimal Blob IDs. The display of Blob output can be controlled using SET BLOBDISPLAY.

Example Suppose that Blob ID 58:c59 refers to graphical data in JPEG format. To place this Blob data into a graphics file named **picture.jpg**, enter:

```
BLOBDUMP 58:c59 picture.jpg;
```

See Also **SET BLOBDISPLAY**

EDIT

Allows editing and re-execution of **isql** commands.

Syntax

```
EDIT [filename];
```

Argument	Description
filename	Name of the file to edit

Description The EDIT command enables you to edit commands in:

- A source file and then execute the commands upon exiting the editor.
- The current **isql** session, then re-execute them.

On Windows platforms, EDIT calls the text editor specified by the EDITOR environment variable. If this environment variable is not defined, then EDIT uses the Microsoft mep editor.

On UNIX, EDIT calls the text editor specified by either the VISUAL environment variable or EDITOR, in that order. If neither variable is defined, then EDIT uses the **vi** editor.

If given filename as an argument, EDIT places the contents of filename in an edit buffer. If no file name is given, EDIT places the commands in the current isql session in the edit buffer.

After exiting the editor, isql automatically executes the commands in the edit buffer.

Filenames with spaces You can optionally delimit the filename with double or single quotes. This allows you to use filenames with spaces in EDIT statements.

Examples To edit the commands in a file called **start.sql** and execute the commands when done, enter:

```
EDIT START.SQL;
```

In the next example, a user wants to enter `SELECT DISTINCT JOB_CODE, JOB_TITLE FROM JOB;` interactively: Instead, the user mistakenly omits the DISTINCT keyword. Issuing the `EDIT` command opens the statement in an editor and then executes the edited statement when the editor exits.

```
SELECT JOB_CODE, JOB_TITLE FROM JOB;
EDIT;
```

See Also **INPUT**, **OUTPUT**, **SHELL**

EXIT

Commits the current transaction, closes the database, and ends the **isql** session.

Syntax

```
EXIT;
```

Description Both EXIT and QUIT close the database and end an **isql** session. EXIT commits any changes made since the last COMMIT or ROLLBACK, whereas QUIT rolls them back.

EXIT is equivalent to the end-of-file character, which differs across systems.

IMPORTANT EXIT commits changes without prompting for confirmation. Before using EXIT, be sure that no transactions need to be rolled back.

See Also **QUIT**, **SET AUTODDL**

HELP

Displays a list of ISQL commands and short descriptions.

Syntax

```
HELP;
```

Description HELP lists the built-in **isql** commands, with a brief description of each.

Example To save the HELP screen to a file named **isqlhelp.lst**, enter:

```
OUTPUT isqlhelp.lst;
HELP;
OUTPUT;
```

After issuing the HELP command, use OUTPUT to redirect output back to the screen.

INPUT

Read and execute commands from the named file.

Syntax

```
INPUT filename;\
```

Argument	Description
filename	Name of the file containing SQL statements and SQL commands

Description INPUT reads commands from *filename* and executes them as a block. In this way, INPUT enables execution of commands without prompting. *filename* must contain SQL statements or **isql** commands.

Input files can contain their own INPUT commands. Nesting INPUT commands enables **isql** to process multiple files. When **isql** reaches the end of one file, processing returns to the previous file until all commands are executed.

The INPUT command is intended for noninteractive use. Therefore, the EDIT command does not work in input files.

Using INPUT *filename* from within an **isql** session has the same effect as using **-input** *filename* from the command line.

Unless output is redirected using OUTPUT, any results returned by executing filename appear on the screen.

You can optionally delimit the filename with double or single quotes. This allows you to use filenames with spaces in INPUT statements.

Examples For this example, suppose that file **add.lst** contains the following INSERT statement:

```
INSERT INTO COUNTRY (COUNTRY, CURRENCY)
VALUES ('Mexico', 'Peso');
```

To execute the command stored in **add.lst**, enter:

```
INPUT add.lst;
```

For the next example, suppose that the file, **table.lst**, contains the following SHOW commands:

```
SHOW TABLE COUNTRY;
SHOW TABLE CUSTOMER;
SHOW TABLE DEPARTMENT;
SHOW TABLE EMPLOYEE;
SHOW TABLE EMPLOYEE_PROJECT;
SHOW TABLE JOB;
```

To execute these commands, enter:

```
INPUT table.lst;
```

To record each command and store its results in a file named **table.out**, enter

```
SET ECHO ON;
OUTPUT table.out;
INPUT table.lst;
OUTPUT;
```

See Also **OUTPUT**

OUTPUT

Redirects output to the named file or to standard output.

Syntax

```
OUTPUT [filename];
```

Argument	Description
filename	Name of the file in which to save output; if no file name is given, results appear on the standard output

Description OUTPUT determines where the results of **isql** commands are displayed. By default, results are displayed on standard output (usually a screen). To store results in a file, supply a *filename* argument. To return to the default mode, again displaying results on the standard output, use OUTPUT without specifying a file name.

By default, only data is redirected. Interactive commands are not redirected unless SET ECHO is in effect. If SET ECHO is in effect, **isql** displays each command before it is executed. In this way, **isql** captures both the results and the command that produced them. SET ECHO is useful for displaying the text of a query immediately before the results.

Note Error messages cannot be redirected to an output file.

Using OUTPUT *filename* from within an **isql** session has the same effect as using the option -**output** *filename* from the command line.

You can optionally delimit the filename with double or single quotes. This allows you to use filenames with spaces in OUTPUT statements.

Example The following example stores the results of one SELECT statement in the file, **sales.out**. Normal output processing resumes after the SELECT statement.

```
OUTPUT sales.out;
SELECT * FROM SALES;
OUTPUT;
```

See Also **INPUT**, **SET ECHO**

QUIT

Rolls back the current transaction, closes the database, and ends the **isql** session.

Syntax

```
QUIT;
```

Description Both EXIT and QUIT close the database and end an **isql** session. QUIT rolls back any changes made since the last COMMIT or ROLLBACK, whereas EXIT commits the changes.

IMPORTANT QUIT rolls back uncommitted changes without prompting for confirmation. Before using QUIT, be sure that any changes that need to be committed are committed. For example, if SET AUTODDL is off, DDL statements must be committed explicitly.

See Also **EXIT**, **SET AUTODDL**

SET

Lists the status of the features that control an **isql** session.

Syntax

```
SET;
```

Description **isql** provides several SET commands for specifying how data is displayed or how other commands are processed.

The SET command, by itself, verifies which features are currently set. Some SET commands turn a feature on or off. Other SET commands assign values.

Many **isql** SET commands have corresponding SQL statements that provide similar or identical functionality. In addition, some of the **isql** features controlled by SET commands can also be controlled using **isql** command-line options. SET Statements are used to configure the **isql** environment from a script file. Changes to the session setting from SET statements in a script affect the session only while the script is running. After a script completes, the session settings prior to running the script are restored.

The **isql** SET statements are:

Statement	**Description**	**Default**
SET AUTODDL	Toggles the commit feature for DDL statements	ON
SET BLOBDISPLAY *n*	Turns on the display of Blob type *n*; the parameter *n* is required to display Blob types	OFF
SET COUNT	Toggles the count of selected rows on or off	OFF
SET ECHO	Toggles the display of each command on or off	OFF
SET LIST *string*	Displays columns vertically or horizontally	OFF

TABLE 9.12 SET statements

Statement	Description	Default
SET NAMES	Specifies the active character set	OFF
SET PLAN	Specifies whether or not to display the optimizer's query plan	OFF
SET STATS	Toggles the display of performance statistics on or off	OFF
SET TERM *string*	Allows you to change to an alternate terminator character	;
SET TIME	Toggles display of time in DATE values	ON

TABLE 9.12 SET statements (*continued*)

By default, all settings are initially OFF except AUTODDL and TIME, and the terminator is a semicolon (**;**). Each time you start an **isql** session or execute an **isql** script file, settings begin with their default values.

SET statements are used to configure the **isql** environment from a script file. Changes to the session setting from SET statements in a script affect the session only while the script is running. After a script completes, the session settings prior to running the script are restored to their values before the script was run. So you can modify the settings for interactive use, then change them as needed in an **isql** script, and after running the script they automatically return to their previous configuration.

Notes

- You cannot enter **isql** SET statements interactively in the SQL Statement area of IBConsole ISQL. You can perform the same functions with menu items.
- SET GENERATOR and SET TRANSACTION (without a transaction name) are DSQL statements and so you can enter them interactively in IBConsole ISQL or **isql**. These statements are not exclusively **isql** statements, so they are not documented in this chapter. See the *Language Reference* for details.
- SET DATABASE is exclusively an embedded SQL statement. See the *Language Reference* and the *Embedded SQL Guide* for details.

Example To display the **isql** features currently in effect, enter:

```
SET;
   Print statistics:OFF
   Echo commands:   OFF
   List format:     OFF
```

```
Row count:       OFF
Autocommit DDL:  OFF
Access plan:     OFF
Display BLOB type:1
Terminator:      ;
Time:            OFF
```

The output shows that **isql** is set to not echo commands, to display Blob data if they are of subtype 1 (text), to automatically commit DDL statements, and to recognize a semicolon (**;**) as the statement termination character.

See Also **SET AUTODDL**, **SET BLOBDISPLAY**, **SET COUNT**, **SET ECHO**, **SET LIST**, **SET NAMES**, **SET PLAN**, **SET STATS**, **SET TERM**, **SET TIME**

SET AUTODDL

Specifies whether DDL statements are committed automatically after being executed or committed only after an explicit COMMIT.

Syntax

```
SET AUTODDL [ON | OFF];
```

Argument	Description
ON	Turns on automatic commitment of DDL [default]
OFF	Turns off automatic commitment of DDL

Description SET AUTODDL is used to turn on or off the automatic commitment of data definition language (DDL) statements. By default, DDL statements are automatically committed immediately after they are executed, in a separate transaction. This is the recommended behavior.

If the OFF keyword is specified, auto-commit of DDL is then turned off. In OFF mode, DDL statements can only be committed explicitly through a user's transaction. This mode is useful for database prototyping, because uncommitted changes are easily undone by rolling them back.

SET AUTODDL has a shorthand equivalent, SET AUTO.

Tip The ON and OFF keywords are optional. If they are omitted, SET AUTO switches from one mode to the other. Although you can save typing by omitting the optional keyword, including the keyword is recommended because it avoids potential confusion.

Examples The following example shows part of an **isql** script that turns off AUTODDL, creates a table named TEMP, then rolls back the work.

```
. . .
SET AUTO OFF;
CREATE TABLE TEMP (a INT, b INT);
ROLLBACK;
. . .
```

This script creates TEMP and then rolls back the statement. No table is created. because its creation was rolled back.

The next script uses the default AUTODDL ON. It creates the table TEMP and then performs a rollback:

```
. . .
CREATE TABLE TEMP (a INT, b INT);
ROLLBACK;
. . .
```

Because DDL is automatically committed, the rollback does not affect the creation of TEMP.

See Also **EXIT**, **QUIT**

SET BLOBDISPLAY

Specifies subtype of Blob data to display.

Syntax `SET BLOBDISPLAY [n | ALL | OFF];`

Argument	Description
n	Integer specifying the Blob subtype to display • Use 0 for Blob data of an unknown subtype • Use 1 for Blob data of a text subtype [default] • Use other integer values for other subtypes
ALL	Displays Blob data of all subtypes
OFF	Turns off display of Blob data of all subtypes

Description SET BLOBDISPLAY has the following uses:

- To display Blob data of a particular subtype, use SET BLOBDISPLAY *n*. By default, **isql** displays Blob data of text subtype (*n* = 1).
- To display Blob data of all subtypes, use SET BLOBDISPLAY ALL.
- To avoid displaying Blob data, use SET BLOBDISPLAY OFF. Omitting the OFF keyword has the same effect. Turn Blob display off to make output easier to read.

In any column containing Blob data, the actual data does not appear in the column. Instead, the column displays a Blob ID that represents the data. If SET BLOBDISPLAY is on, data associated with a Blob ID appears under the row containing the Blob ID. If SET BLOBDISPLAY is off, the Blob ID still appears even though its associated data does not.

SET BLOBDISPLAY has a shorthand equivalent, SET BLOB.

To determine the subtype of a BLOB column, use SHOW TABLE.

Examples

The following examples show output from the same SELECT statement. Each example uses a different SET BLOB command to affect how output appears. The first example turns off Blob display.

```
SET BLOB OFF;
SELECT PROJ_NAME, PROJ_DESC FROM PROJECT;
```

With BLOBDISPLAY OFF, the output shows only the Blob ID:

```
PROJ_NAME              PROJ_DESC
====================   =================
Video Database         24:6
DigiPizza              24:8
AutoMap                24:a
MapBrowser port        24:c
Translator upgrade     24:3b
Marketing project 3    24:3d
```

The next example restores the default by setting BLOBDISPLAY to subtype 1 (text).

```
SET BLOB 1;
SELECT PROJ_NAME, PROJ_DESC FROM PROJECT;
```

Now the contents of the Blob appear below each Blob ID:

```
PROJ_NAME        PROJ_DESC
=====================================
Video Database   24:6
==============================================================
PROJ_DESC:
Design a video data base management system for
controlling on-demand video distribution.
```

```
PROJ_NAME        PROJ_DESC
=====================================
DigiPizza        24:8
==============================================================
PROJ_DESC:
Develop second generation digital pizza maker
with flash-bake heating element and
digital ingredient measuring system.
. . .
```

See Also **BLOBDUMP**

SET COUNT

Specifies whether to display number of rows retrieved by queries.

Syntax

```
SET COUNT [ON | OFF];
```

Argument	Description
ON	Turns on display of the “rows returned” message
OFF	Turns off display of the “rows returned” message [default]

Description By default, when a SELECT statement retrieves rows from a query, no message appears to say how many rows were retrieved.

Use SET COUNT ON to change the default behavior and display the message. To restore the default behavior, use SET COUNT OFF.

Tip The ON and OFF keywords are optional. If they are omitted, SET COUNT switches from one mode to the other. Although you can save typing by omitting the optional keyword, including the keyword is recommended because it avoids potential confusion.

Example The following example sets COUNT ON to display the number of rows returned by all following queries:

```
SET COUNT ON;
SELECT * FROM COUNTRY
   WHERE CURRENCY LIKE '%FRANC%';
```

The output displayed would then be:

```
   COUNTRY          CURRENCY
   ===============  ==========
   SWITZERLAND      SFRANC
   FRANCE           FFRANC
   BELGIUM          BFRANC
3 rows returned
```

SET ECHO

Specifies whether commands are displayed to the **isql** Output area before being executed.

Syntax `SET ECHO [ON | OFF];`

Argument	Description
ON	Turns on command echoing [default]
OFF	Turns off command echoing

Description By default, commands in script files are displayed (echoed) in the **isql** Output area, before being executed. Use SET ECHO OFF to change the default behavior and suppress echoing of commands. This can be useful when sending the output of a script to a file, if you want only the results of the script and not the statements themselves in the output file.

Command echoing is useful if you want to see the commands as well as the results in the **isql** Output area.

Tip The ON and OFF keywords are optional. If they are omitted, SET ECHO switches from one mode to the other. Although you can save typing by omitting the optional keyword, including the keyword is recommended because it avoids potential confusion.

Example Suppose you execute the following script from IBConsole ISQL:

```
. . .
SET ECHO OFF;
SELECT * FROM COUNTRY;
SET ECHO ON;
SELECT * FROM COUNTRY;
EXIT;
```

The output (in a file or the **isql** Output area) looks like this:

```
. . .
SET ECHO OFF;
COUNTRY         CURRENCY
```

```
===========   ========
USA           Dollar
England       Pound
. . .
SELECT * FROM COUNTRY;
COUNTRY       CURRENCY
===========   ========
USA           Dollar
England       Pound
. . .
```

The first SELECT statement is not displayed, because ECHO is OFF. Notice also that the SET ECHO ON statement itself is not displayed, because when it is executed, ECHO is still OFF. After it is executed, however, the second SELECT statement is displayed.

See Also **INPUT**, **OUTPUT**

SET LIST

Specifies whether output appears in tabular format or in list format.

Syntax

```
SET LIST [ON | OFF];
```

Argument	Description
ON	Turns on list format for display of output
OFF	Turns off list format for display of output [default]

Description By default, when a SELECT statement retrieves rows from a query, the output appears in a tabular format, with data organized in rows and columns.

Use SET LIST ON to change the default behavior and display output in a list format. In list format, data appears one value per line, with column headings appearing as labels. List format is useful when columnar output is too wide to fit nicely on the screen.

Tip The ON and OFF keywords are optional. If they are omitted, SET LIST switches from one mode to the other. Although you can save typing by omitting the optional keyword, including the keyword is recommended because it avoids potential confusion.

Example Suppose you execute the following statement in a script file:

```
SELECT JOB_CODE, JOB_GRADE, JOB_COUNTRY, JOB_TITLE FROM JOB
    WHERE JOB_COUNTRY = 'Italy';
```

The output is:

```
JOB_CODE   JOB_GRADE   JOB_COUNTRY   JOB_TITLE
========   =========   ===========   ====================
SRep       4           Italy         Sales Representative
```

Now suppose you precede the SELECT with SET LIST ON:

```
SET LIST ON;
SELECT JOB_CODE, JOB_GRADE, JOB_COUNTRY, JOB_TITLE FROM JOB
   WHERE JOB_COUNTRY = 'Italy';
```

The output is:

```
   JOB_CODE        SRep
   JOB_GRADE       4
   JOB_COUNTRY     Italy
   JOB_TITLE       Sales Representative
```

SET NAMES

Specifies the active character set to use in database transactions.

Syntax

```
SET NAMES [charset];
```

Argument	**Description**
charset	Name of the active character set; default is NONE

Description SET NAMES specifies the character set to use for subsequent database connections in **isql**. It enables you to override the default character set for a database. To return to using the default character set, use SET NAMES with no argument.

Use SET NAMES before connecting to the database whose character set you want to specify. For a complete list of character sets recognized by InterBase, see the *Language Reference*.

Choice of character set limits possible collation orders to a subset of all available collation orders. Given a specific character set, a specific collation order can be specified when data is selected, inserted, or updated in a column.

Example The following statement at the beginning of a script file indicates to set the active character set to ISO8859_1 for the subsequent database connection:

```
SET NAMES ISO8859_1;
CONNECT 'jupiter:/usr/interbase/examples/employee.gdb';
. . .
```

SET PLAN

Specifies whether to display the optimizer's query plan.

Syntax

```
SET PLAN [ON | OFF];
```

Argument	Description
ON	Turns on display of the optimizer's query plan
OFF	Turns off display of the optimizer's query plan [default]

Description By default, when a SELECT statement retrieves rows from a query, **isql** does not display the query plan used to retrieve the data.

Use SET PLAN ON to change the default behavior and display the query optimizer plan. To restore the default behavior, use SET PLAN OFF.

To change the query optimizer plan, use the PLAN clause in the SELECT statement.

Tip The ON and OFF keywords are optional. If they are omitted, SET PLAN switches from one mode to the other. Although you can save typing by omitting the optional keyword, including the keyword is recommended because it avoids potential confusion.

Example The following example shows part of a script that sets PLAN ON:

```
SET PLAN ON;
SELECT JOB_COUNTRY, MIN_SALARY FROM JOB
   WHERE MIN_SALARY > 50000
      AND JOB_COUNTRY = 'France';
```

The output then includes the query optimizer plan used to retrieve the data as well as the results of the query:

```
PLAN (JOB INDEX (RDB$FOREIGN3,MINSALX,MAXSALX))
JOB_COUNTRY      MIN_SALARY
===============  ======================
France           118200.00
```

SET STATS

Specifies whether to display performance statistics after the results of a query.

Syntax

```
SET STATS [ON | OFF];
```

Argument	Description
ON	Turns on display of performance statistics
OFF	Turns off display of performance statistics [default]

Description By default, when a SELECT statement retrieves rows from a query, **isql** does not display performance statistics after the results. Use SET STATS ON to change the default behavior and display performance statistics. To restore the default behavior, use SET STATS OFF. Performance statistics include:

- Current memory available, in bytes
- Change in available memory, in bytes
- Maximum memory available, in bytes
- Elapsed time for the operation
- CPU time for the operation
- Number of cache buffers used
- Number of reads requested
- Number of writes requested
- Number of fetches made

Performance statistics can help determine if changes are needed in system resources, database resources, or query optimization.

Tip The ON and OFF keywords are optional. If they are omitted, SET STATS switches from one mode to the other. Although you can save typing by omitting the optional keyword, including the keyword is recommended because it avoids potential confusion.

Do not confuse SET STATS with the SQL statement SET STATISTICS, which recalculates the selectivity of an index.

Example The following part of a script file turns on display of statistics and then performs a query:

```
SET STATS ON;
SELECT JOB_COUNTRY, MIN_SALARY FROM JOB
    WHERE MIN_SALARY > 50000
        AND JOB_COUNTRY = 'France';
```

The output displays the results of the SELECT statement and the performance statistics for the operation:

```
    JOB_COUNTRY      MIN_SALARY
    ===============  ======================
    France           118200.00

Current memory = 407552
Delta memory = 0
Max memory = 412672
Elapsed time= 0.49 sec
Cpu = 0.06 sec
Buffers = 75
Reads = 3
Writes = 2
Fetches = 441
```

See Also **SHOW DATABASE**

SET TERM

Specifies which character or characters signal the end of a command.

Syntax `SET TERM string;`

Argument	Description
string	Specifies a character or characters to use in terminating a statement; default is semicolon (**;**)

Description By default, when a line ends with a semicolon, **isql** interprets it as the end of a command. Use SET TERM to change the default behavior and define a new termination character.

SET TERM is typically used with CREATE PROCEDURE or CREATE TRIGGER. Procedures and triggers are defined using a special “procedure and trigger language” in which statements end with a semicolon. If **isql** were to interpret semicolons as statement terminators, then procedures and triggers would execute during their creation, rather than when they are called.

A script file containing CREATE PROCEDURE or CREATE TRIGGER definitions should include one SET TERM command before the definitions and a corresponding SET TERM after the definitions. The beginning SET TERM defines a new termination character; the ending SET TERM restores the semicolon (**;**) as the default.

Note You do not need to change the terminator before entering an interactive CREATE PROCEDURE or CREATE TRIGGER statement in the IBConsole ISQL SQL statement area. The contents of the SQL statement area is always treated as one DSQL statement, even if it contains semicolons. Use of SET TERM is necessary only in command-line **isql** and when running SQL script files from command-line **isql** or IBConsole ISQL.

Example The following example shows a text file that uses SET TERM in creating a procedure. The first SET TERM defines "##" as the termination characters. The matching SET TERM restores ";" as the termination character.

```
SET TERM ## ;
CREATE PROCEDURE ADD_EMP_PROJ (EMP_NO SMALLINT, PROJ_ID CHAR(5))
AS
BEGIN
   BEGIN
      INSERT INTO EMPLOYEE_PROJECT (EMP_NO, PROJ_ID)
         VALUES (:emp_no, :proj_id);
   WHEN SQLCODE -530 DO
   EXCEPTION UNKNOWN_EMP_ID;
   END
   RETURN;
END ##
SET TERM ; ##
```

SET TIME

Specifies whether to display the time portion of a DATE value.

Syntax

```
SET TIME [ON | OFF];
```

Argument	Description
ON	Turns on display of time in DATE value
OFF	Turns off display of time in DATE value [default]

Description The InterBase Date datatype includes a date portion (including day, month, and year) and a time portion (including hours, minutes, and seconds).

By default, **isql** displays only the date portion of Date values. SET TIME ON turns on the display of time values. SET TIME OFF turns off the display of time values.

Tip The ON and OFF keywords are optional. If they are omitted, the command toggles time display from ON to OFF or OFF to ON.

Example The following example shows the default display of a DATE datatype, which is to display day, month, and year:

```
SELECT HIRE_DATE FROM EMPLOYEE WHERE EMP_NO = 145;
HIRE_DATE
-------------------
2-MAY-1994
```

This example shows the effects of SET TIME ON, which causes the hours, minutes and seconds to be displayed as well:

```
SET TIME ON;
SELECT HIRE_DATE FROM EMPLOYEE WHERE EMP_NO = 145;
HIRE_DATE
-------------------
2-MAY-1994 12:25:00
```

SHELL

Allows execution of an operating system command or temporary access to an operating system shell.

Syntax
```
SHELL [<os_command>];
```

Argument	**Description**
os_command	An operating system command; if no command is specified, **isql** provides interactive access to the operating system

Description The SHELL command provides temporary access to operating system commands in an **isql** session. Use SHELL to execute an operating-system command without ending the current **isql** session.

If *os_command* is specified, the operating system executes the command and then returns to **isql** when complete.

If no command is specified, an operating system shell prompt appears, enabling you to execute a sequence of commands. To return to **isql**, type `exit`. For example, SHELL can be used to edit an input file and run it at a later time. By contrast, if an input file is edited using the EDIT command, the input file is executed as soon as the editing session ends.

Using SHELL does not commit transactions before it calls the shell.

This **isql** statement has no equivalent function in IBConsole ISQL.

Example The following example uses SHELL to display the contents of the current directory:

```
SHELL DIR;
```

See Also **EDIT**

SHOW CHECK

Displays all CHECK constraints defined for a specified table.

Syntax

```
SHOW CHECK table;
```

Argument	Description
table	Name of an existing table in the current database

Description SHOW CHECK displays CHECK constraints for a named table in the current database. Only user-defined metadata is displayed. To see a list of existing tables, use SHOW TABLE.

Example The following example shows CHECK constraints defined for the JOB table. The SHOW TABLES command is used first to display a list of available tables.

```
SHOW TABLES;
   COUNTRY          CUSTOMER
   DEPARTMENT       EMPLOYEE
   EMPLOYEE_PROJECT JOB
   PHONE_LIST       PROJECT
   PROJ_DEPT_BUDGET SALARY_HISTORY
   SALES

SHOW CHECK JOB;
   CHECK (min_salary < max_salary)
```

See Also **SHOW TABLES**

SHOW DATABASE

Displays information about the current database.

Syntax

```
SHOW [DATABASE | DB];
```

Description SHOW DATABASE displays the current database's file name, page size and allocation, and sweep interval.

The output of SHOW DATABASE is used to verify data definition or to administer the database. For example, use the backup and restore utilities to change page size or reallocate pages among multiple files, and use the database maintenance utility to change the sweep interval.

SHOW DATABASE has a shorthand equivalent, SHOW DB.

Example The following example connects to a database and displays information about it:

```
CONNECT 'employee.gdb';
   Database: employee.gdb

SHOW DB;
   Database: employee.gdb
         Owner: SYSDBA
   PAGE_SIZE 4096
   Number of DB pages allocated = 422
   Sweep interval = 20000
```

SHOW DOMAINS

Lists all domains or displays information about a specified domain.

Syntax

```
SHOW {DOMAINS | DOMAIN name};
```

Argument	Description
name	Name of an existing domain in the current database

Options To see a list of existing domains, use SHOW DOMAINS without specifying a domain name. SHOW DOMAIN name displays information about the named domain in the current database. Output includes a domain's datatype, default value, and any CHECK constraints defined. Only user-defined metadata is displayed.

Example The following example lists all domains and then shows the definition of the domain, SALARY:

```
SHOW DOMAINS;
   FIRSTNAME        LASTNAME
   PHONENUMBER      COUNTRYNAME
   ADDRESSLINE      EMPNO
   DEPTNO           PROJNO
   CUSTNO           JOBCODE
   JOBGRADE         SALARY
   BUDGET           PRODTYPE
   PONUMBER

SHOW DOMAIN SALARY;
   SALARY           NUMERIC(15, 2) Nullable
                    DEFAULT 0
                    CHECK (VALUE > 0)
```

SHOW EXCEPTIONS

Lists all exceptions or displays the text of a specified exception.

Syntax `SHOW {EXCEPTIONS | EXCEPTION name};`

Argument	Description
name	Name of an existing exception in the current database

Description SHOW EXCEPTIONS displays an alphabetical list of exceptions. SHOW EXCEPTION name displays the text of the named exception.

Examples To list all exceptions defined for the current database, enter:

```
SHOW EXCEPTIONS;
Exception Name   Used by, Type
================== ========================================
UNKNOWN_EMP_ID   ADD_EMP_PROJ, Stored procedure
   Invalid employee number or project ID.
. . .
```

To list the message for a specific exception and the procedures or triggers that use it, enter the exception name:

```
SHOW EXCEPTION CUSTOMER_CHECK;
Exception Name                   Used by, Type
===========================  =======================================
CUSTOMER_CHECK               SHIP_ORDER, Stored procedure
 Overdue balance -- can’t ship.
```

SHOW FILTERS

Lists all Blob filters or displays information about a specified filter.

Syntax

```
SHOW {FILTERS | FILTER name};
```

Argument	Description
name	Name of an existing Blob filter in the current database

Options To see a list of existing filters, use SHOW FILTERS. SHOW FILTER name displays information about the named filter in the current database. Output includes information previously defined by the DECLARE FILTER statement, the input subtype, output subtype, module (or library) name, and entry point name.

Example The following example lists all filters and then shows the definition of the filter, DESC_FILTER:

```
SHOW FILTERS;
   DESC_FILTER

SHOW FILTER DESC_FILTER;
   BLOB Filter: DESC_FILTER
   Input subtype: 1 Output subtype -4
   Filter library is: desc_filter
   Entry point is: FILTERLIB
```

SHOW FUNCTIONS

Lists all user-defined functions (UDFs) defined in the database or displays information about a specified UDF.

Syntax

```
SHOW {FUNCTIONS | FUNCTION name};
```

Argument	Description
name	Name of an existing UDF in the current database

Options To see a list of existing functions defined in the database, use SHOW FUNCTIONS. SHOW FUNCTION name displays information about the named function in the current database. Output includes information previously defined by the DECLARE EXTERNAL FUNCTION statement: the name of the function and function library, the name of the entry point, and the datatypes of return values and input arguments.

Example The following example lists all UDFs and then shows the definition of the MAXNUM() function:

```
SHOW FUNCTIONS;
   ABS       MAXNUM
   TIME      UPPER_NON_C
   UPPER

SHOW FUNCTION maxnum;
   Function MAXNUM:
   Function library is /usr/interbase/lib/gdsfunc.so
   Entry point is FN_MAX
   Returns BY VALUE DOUBLE PRECISION
   Argument 1: DOUBLE PRECISION
   Argument 2: DOUBLE PRECISION
```

SHOW GENERATORS

Lists all generators or displays information about a specified generator.

Syntax `SHOW {GENERATORS | GENERATOR name};`

Argument	Description
name	Name of an existing generator in the current database

Description To see a list of existing generators, use SHOW GENERATORS. SHOW GENERATOR name displays information about the named generator in the current database. Output includes the name of the generator and its next value.

SHOW GENERATOR has a shorthand equivalent, SHOW GEN.

Example The following example lists all generators and then shows information about EMP_NO_GEN:

```
SHOW GENERATORS;
   Generator EMP_NO_GEN, Next value: 146
   Generator CUST_NO_GEN, Next value: 1016
```

```
SHOW GENERATOR EMP_NO_GEN;
   Generator EMP_NO_GEN, Next value: 146
```

SHOW GRANT

Displays privileges for a database object.

Syntax

```
SHOW GRANT object;
```

Argument	Description
object	Name of an existing table, view, or procedure in the current database

Description SHOW GRANT displays the privileges defined for a specified table, view, or procedure. Allowed privileges are DELETE, EXECUTE, INSERT, SELECT, UPDATE, or ALL. To change privileges, use the SQL statements GRANT or REVOKE.

Before using SHOW GRANT, you might want to list the available database objects. Use SHOW PROCEDURES to list existing procedures; use SHOW TABLES to list existing tables; use SHOW VIEWS to list existing views.

Example To display GRANT privileges on the JOB table, enter:

```
SHOW GRANT JOB;
   GRANT SELECT ON JOB TO ALL
   GRANT DELETE, INSERT, SELECT, UPDATE ON JOB TO MANAGER
```

SHOW GRANT can also show role membership:

```
SHOW GRANT DOITALL;
   GRANT DOITALL TO SOCKS
```

See Also **SHOW PROCEDURES**, **SHOW TABLES**, **SHOW VIEWS**

SHOW INDEX

Displays index information for a specified index, for a specified table, or for all tables in the current database.

Syntax

```
SHOW {INDICES | INDEX {index | table} };
```

Argument	Description
index	Name of an existing index in the current database
table	Name of an existing table in the current database

Description SHOW INDEX displays the index name, the index type (for example, UNIQUE or DESC), and the columns on which an index is defined.

If the index argument is specified, SHOW INDEX displays information only for that index. If table is specified, SHOW INDEX displays information for all indexes in the named table; to display existing tables, use SHOW TABLES. If no argument is specified, SHOW INDEX displays information for all indexes in the current database.

SHOW INDEX has a shorthand equivalent, SHOW IND. SHOW INDICES is also a synonym for SHOW INDEX. SHOW INDEXES is not supported.

Examples To display indexes for database **employee.gdb**, enter:

```
SHOW INDEX;
   RDB$PRIMARY1 UNIQUE INDEX ON COUNTRY(COUNTRY)
   CUSTNAMEX INDEX ON CUSTOMER(CUSTOMER)
   CUSTREGION INDEX ON CUSTOMER(COUNTRY, CITY)
   RDB$FOREIGN23 INDEX ON CUSTOMER(COUNTRY)
. . .
```

To display index information for the SALES table, enter:

```
SHOW IND SALES;
   NEEDX INDEX ON SALES(DATE_NEEDED)
   QTYX DESCENDING INDEX ON SALES(ITEM_TYPE, QTY_ORDERED)
   RDB$FOREIGN25 INDEX ON SALES(CUST_NO)
   RDB$FOREIGN26 INDEX ON SALES(SALES_REP)
   RDB$PRIMARY24 UNIQUE INDEX ON SALES(PO_NUMBER)
   SALESTATX INDEX ON SALES(ORDER_STATUS, PAID)
```

See Also **SHOW TABLES**

SHOW PROCEDURES

Lists all procedures or displays the text of a specified procedure.

Syntax

```
SHOW {PROCEDURES | PROCEDURE name};
```

Argument	Description
name	Name of an existing procedure in the current database

Description SHOW PROCEDURES displays an alphabetical list of procedures, along with the database objects they depend on. Deleting a database object that has a dependent procedure is not allowed. To avoid an **isql** error, delete the procedure (using DROP PROCEDURE) before deleting the database object.

SHOW PROCEDURE name displays the text and parameters of the named procedure.

SHOW PROCEDURE has a shorthand equivalent, SHOW PROC.

Examples To list all procedures defined for the current database, enter:

```
SHOW PROCEDURES;

    Procedure Name   Dependency Type
    ================= ==================== =======
    ADD_EMP_PROJ     EMPLOYEE_PROJECTTable
                     UNKNOWN_EMP_IDException
    DELETE_EMPLOYEE  DEPARTMENTTable
                     EMPLOYEETable
                     EMPLOYEE_PROJECTTable
                     PROJECTTable
                     REASSIGN_SALESException
                     SALARY_HISTORYTable
                     SALES  Table
    DEPT_BUDGET      DEPARTMENTTable
                     DEPT_BUDGETProcedure
. . .
```

To display the text of the procedure, ADD_EMP_PROJ, enter:

```
SHOW PROC ADD_EMP_PROJ;

    Procedure text:
    =================================================================

    BEGIN
        BEGIN
        INSERT INTO EMPLOYEE_PROJECT (EMP_NO, PROJ_ID) VALUES (:emp_no,
            :proj_id);
        WHEN SQLCODE -530 DO
        EXCEPTION UNKNOWN_EMP_ID;
```

```
      END
      RETURN;
   END
   ==================================================================
   Parameters:
   EMP_NO INPUT SMALLINT
   PROJ_ID INPUT CHAR(5)
```

SHOW ROLES

Displays the names of SQL roles for the current database.

Syntax `SHOW {ROLES | ROLE}`

Description SHOW ROLES displays the names of all roles defined for the current database. To show user membership in roles, use SHOW GRANT *rolename*.

Example

```
SHOW ROLES;
   DOITALL       DONOTHING
   DOONETHING    DOSOMETHING
```

See Also **SHOW GRANT**

SHOW SYSTEM

Displays the names of system tables and system views for the current database.

Syntax `SHOW SYSTEM [TABLES];`

Description SHOW SYSTEM lists system tables and system views in the current database. SHOW SYSTEM accepts an optional keyword, TABLES, which does not affect the behavior of the command.

SHOW SYSTEM has a shorthand equivalent, SHOW SYS.

Example To list system tables and system views for the current database, enter:

```
SHOW SYS;
   RDB$CHARACTER_SETS       RDB$CHECK_CONSTRAINTS
   RDB$COLLATIONS           RDB$DATABASE
   RDB$DEPENDENCIES         RDB$EXCEPTIONS
   RDB$FIELDS               RDB$FIELD_DIMENSIONS
   RDB$FILES                RDB$FILTERS
   RDB$FORMATS            RDB$FUNCTIONS
   RDB$FUNCTION_ARGUMENTS RDB$GENERATORS
   RDB$INDEX_SEGMENTS     RDB$INDICES
   RDB$LOG_FILES          RDB$PAGES
   RDB$PROCEDURES         RDB$PROCEDURE_PARAMETERS
   RDB$REF_CONSTRAINTS    RDB$RELATIONS
   RDB$RELATION_CONSTRAINTS RDB$RELATION_FIELDS
   RDB$ROLES              RDB$SECURITY_CLASSES
   RDB$TRANSACTIONS       RDB$TRIGGERS
   RDB$TRIGGER_MESSAGES   RDB$TYPES
   RDB$USER_PRIVILEGES    RDB$VIEW_RELATIONS
```

See Also For more information about system tables, see the *Language Reference*.

SHOW TABLES

Lists all tables or views, or displays information about a specified table or view.

Syntax

```
SHOW {TABLES | TABLE name};
```

Argument	Description
name	Name of an existing table or view in the current database

Description SHOW TABLES displays an alphabetical list of tables and views in the current database. To determine which listed objects are views rather than tables, use SHOW VIEWS.

SHOW TABLE name displays information about the named object. If the object is a table, command output lists column names and definitions, PRIMARY KEY, FOREIGN KEY, and CHECK constraints, and triggers. If the object is a view, command output lists column names and definitions, as well as the SELECT statement that the view is based on.

Examples To list all tables or views defined for the current database, enter:

```
SHOW TABLES;
   COUNTRY          CUSTOMER
   DEPARTMENT       EMPLOYEE
   EMPLOYEE_PROJECT JOB
   PHONE_LIST       PROJECT
   PROJ_DEPT_BUDGET SALARY_HISTORY
   SALES
```

To show the definition for the COUNTRY table, enter:

```
SHOW TABLE COUNTRY;
   COUNTRY (COUNTRYNAME) VARCHAR(15) NOT NULL
   CURRENCY VARCHAR(10) NOT NULL
   PRIMARY KEY (COUNTRY)
```

See Also **SHOW VIEWS**

SHOW TRIGGERS

Lists all triggers or displays information about a specified trigger.

Syntax

```
SHOW {TRIGGERS | TRIGGER name};
```

Argument	Description
name	Name of an existing trigger in the current database

Description SHOW TRIGGERS displays all triggers defined in the database, along with the table they depend on. SHOW TRIGGER name displays the name, sequence, type, activation status, and definition of the named trigger.

SHOW TRIGGER has a shorthand equivalent, SHOW TRIG.

Deleting a table that has a dependent trigger is not allowed. To avoid an **isql** error, delete the trigger (using DROP TRIGGER) before deleting the table.

Examples To list all triggers defined for the current database, enter:

```
SHOW TRIGGERS;
   Table name      Trigger name
   ===========     ============
   EMPLOYEE        SET_EMP_NO
   EMPLOYEE        SAVE_SALARY_CHANGE
   CUSTOMER        SET_CUST_NO
   SALES           POST_NEW_ORDER
```

To display information about the SET_CUST_NO trigger, enter:

```
SHOW TRIG SET_CUST_NO;

   Triggers:
   SET_CUST_NO, Sequence: 0, Type: BEFORE INSERT, Active
   AS
   BEGIN
```

```
    new.cust_no = gen_id(cust_no_gen, 1);
END
```

SHOW VERSION

Displays information about software versions.

Syntax

```
SHOW VERSION;
```

Description SHOW VERSION displays the software version of **isql**, the InterBase engine, and the on-disk structure (ODS) of the database to which the session is attached.

Certain tasks might not work as expected if performed on databases that were created using older versions of InterBase. To check the versions of software that are running, use SHOW VERSION.

SHOW VERSION has a shorthand equivalent, SHOW VER.

Example To display software versions, enter:

```
SHOW VER;
    ISQL Version: WI-V6.5.5
    InterBase/Windows NT (access method), version 'WI-V6.5.5'
    on disk structure version 9.1
```

See Also **SHOW DATABASE**

SHOW VIEWS

Lists all views or displays information about a specified view.

Syntax

```
SHOW {VIEWS | VIEW name};
```

Argument	Description
name	Name of an existing view in the current database

Description SHOW VIEWS displays an alphabetical list of all views in the current database. SHOW VIEW name displays information about the named view.

Example To list all views defined for the current database, enter:

```
SHOW VIEWS;
    PHONE_LIST
```

See Also **SHOW TABLES**

Using SQL scripts

The basic steps for using script files are:

1. Create the script file using a text editor.
2. Run the file with **isql** or IBConsole.
3. View output and confirm database changes.

Creating an isql script

You can use any text editor to create an SQL script file, as long as the final file format is plain text (ASCII).

Every SQL script file must begin with either a CREATE DATABASE statement or a CONNECT statement (including username and password) that specifies the database on which the script file is to operate. The CONNECT or CREATE statement must contain a complete database file name and directory path.

Note You cannot set dialect in a CREATE DATABASE statement. To create a dialect 3 database, specify **isql** option -s 3.

An SQL script can contain any of the following elements:

- SQL statements, as described in the *Language Reference*
- **isql** SET commands as described in this chapter
- Comments.

Each SQL statement in a script must be terminated by a semicolon (**;**) or the current terminator if it has been changed with SET TERM.

Note The SQL statement silently fails if significant text follows the terminator character on the same line. Whitespace and comments can safely follow the terminator, but other statements cannot.

Each SQL script file should end with either EXIT to commit database changes made since the last COMMIT, or QUIT to roll back changes made by the script. If neither is specified, then database changes are committed by default.

For the full syntax of CONNECT and CREATE DATABASE, see the *Language Reference*.

Running an SQL script

The following steps execute all the SQL statements in the specified script file. The contents of the script are not displayed in the SQL Input Area.

To run a script file containing SQL statements using IBConsole:

1. If you are not already in the SQL window, click the Launch SQL toolbar button or choose **Tools | Interactive SQL**.
2. If you are not running the SQL script on the database to which you are currently connected, then check that the file begins with a valid, uncommented, CONNECT or CREATE DATABASE statement.
3. Choose **Query | Load Script**.
4. Enter or locate the desired script filename in the Open dialog, and click Open to load the script into the SQL input area.
5. Click the Execute toolbar button, or choose **Query | Execute**.

If IBConsole encounters an error, an information dialog appears indicating the error. Once IBConsole finishes executing the script, the script results are displayed in the SQL output window.

After a script executes, all ISQL session settings prior to executing the script are restored as well as the previous database connection, if any. In other words, any **isql** SET commands in the script affect only the **isql** session while the script is running.

Committing work in an SQL script

Changes to the database from data definition (DDL) statements—for example, CREATE and ALTER statements—are automatically committed by default. This means that other users of the database see changes as soon as each DDL statement is executed. To turn off automatic commit of DDL in a script, use SET AUTODDL OFF, or set it in the Query Options dialog. See **"Using InterBase Manager to start and stop InterBase" on page 43** for more information.

Note When creating tables and other database objects with AUTODDL OFF, it is good practice to put a COMMIT statement in the SQL script after each CREATE statement or group of related statements. This ensures that other users of the database see the objects immediately.

Changes made to the database by data manipulation (DML) statements—for example INSERT and UPDATE—are not permanent until they are committed. Commit changes in a script with COMMIT. To undo all database changes since the last COMMIT, use ROLLBACK. For the full syntax of COMMIT and ROLLBACK, see the *Language Reference* book.

Adding comments in an isql script

isql scripts are commented exactly like C programs:

```
/* comment */
```

A comment can occur on the same line as an SQL statement or **isql** command and can be of any length, as long as it is preceded by “/*” and followed by “*/”.

CHAPTER

10

Database and Server Performance

This chapter describes techniques for designing and operating an InterBase client/server system for best speed and efficiency.

The guidelines in this chapter are organized into the following categories:

- Hardware configuration
- Operating system configuration
- Network configuration
- Database properties
- Database design principles
- Database tuning tasks
- Application design techniques
- Application development tools

Introduction

One of the most important requirements for a database as part of your application is to store and retrieve data as quickly as possible. Like any software development technique, there is always more than one method to implement a given specified software solution, and it takes knowledge and experience to choose the design that results in the most efficient operation and the highest performance.

Each project offers unique challenges and requires specific solutions. The suggestions in this chapter augment your own software engineering discipline, which should include careful analysis, testing, and experimentation to implement the best design for your specific project.

Hardware configuration

This section gives guidelines for platform hardware sizing. The suggestions focus on requirements for a server platform.

Choosing a processor speed

The performance of database systems tends by nature to be bound by I/O bandwidth or network bandwidth. An application often waits for I/O or network operations, instead of being computationally intensive. A fast CPU clock speed gives definite performance advantage, but a 10% increase in CPU clock speed is less important for server performance than some other hardware factors, such as RAM configuration, I/O system, or network hardware.

CPU clock speed is often more important on client platforms, because applications that use data might perform CPU-intensive computational analysis on data, or might render sophisticated visualization of data in a computationally costly manner.

It’s not appropriate for this document to recommend a specific CPU clock speed for your server, because it is likely that such a recommendation would be obsolete as you read it. You should evaluate the benefit of spending more money on a faster CPU, because the price/performance curve becomes steep for the latest CPU hardware.

Using multiprocessor servers

With current InterBase SuperServer implementation, you are likely to gain only a modest performance improvement by using multiprocessor hardware. The InterBase SuperServer engine is certified to work on symmetric multiprocessor (SMP) hardware, but doesn't currently implement parallel execution features.

The reason that the multithreaded SuperServer does not take full advantage of SMP configurations is that the InterBase lock manager is a single-threaded section of code. Database requests tend to serialize in order to acquire locks. This usually isn't a severe bottleneck, because lock management is a high-throughput operation, compared to physical I/O.

The InterBase Classic implementation, which executes an individual process on the server for each client connection, benefits more than SuperServer from SMP. However, Classic does not benefit from performance and scalability features that SuperServer provides when the number of simultaneous users grows: the shared data cache and fast interthread concurrency management.

SMP systems do benefit the InterBase server in that additional CPUs can take the load of other processing for the server, such as network services, desktop management, and other application processes. The amount of performance improvement in this case depends on the demands of other processes relative to the InterBase server process. Expect between a 5 and 20 percent performance improvement on a multipurpose server by using multiple processors instead of a single processor.

On a dedicated server, SMP actually tends to decrease performance of InterBase on Windows NT. See **"Understanding Windows NT pitfalls" on page 259**.

Sizing memory

It is important to equip your server with a sufficient amount of physical memory to ensure good performance.

While InterBase can function in a low-profile hardware configuration, with as little as 32MB of RAM on most operating systems, it is recommended to have at least 64MB of RAM on a server system. Database servers that experience a high load can benefit from more RAM.

The base RAM requirement of the **ibserver** executable and for each connected user is low: approximately 1500KB, plus 28KB for each client connection. **ibserver** caches metadata and data for each database to which it connects. User operations such as sorting temporarily consume additional memory. A heavily loaded server with dozens of clients performing concurrent queries requires up to 256MB of RAM.

On Windows NT, you can use the Task Manager, Performance Monitor, and other tools to monitor the resource use of **ibserver**. UNIX and Linux servers have similar resource consumption reporting tools. Add RAM to a system that shows too many page faults.

Using high-performance I/O subsystems

A multiuser database server's hard drives are no place to be thrifty, especially in today's market of inexpensive storage. Configuring a relatively high-end I/O system is a cost-effective way to increase performance.

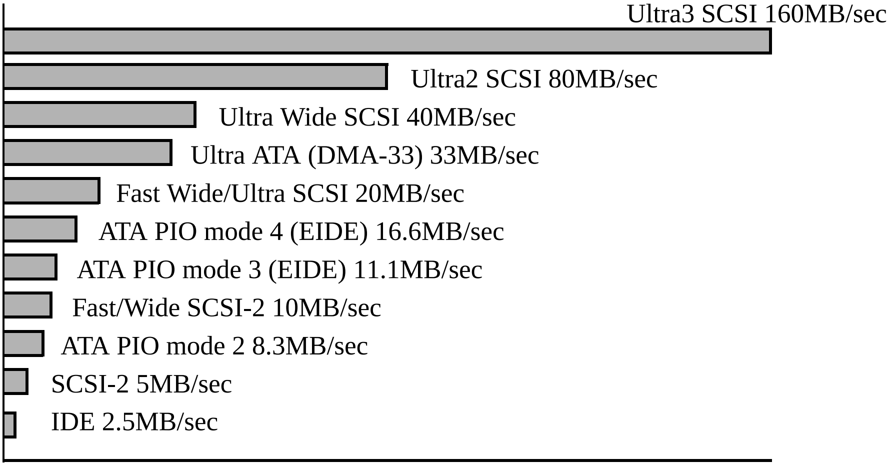
Slow disk subsystems are often the weak link in an otherwise high-performance server machine. The top-rated CPU and maximum memory helps. But if a cheap disk I/O interface limits the data transfer rate, then the money spent on the expensive components is wasted.

It's not appropriate for this document to recommend a particular configuration. The technology changes so quickly that any recommendation here would be outdated. When you specify the machine for a server platform, research the best hardware solution available.

Read the following guidelines for principles:

- Advanced SCSI technology offers superior I/O throughput. The following graph illustrates the relative maximum throughput of different disk interfaces.

FIGURE 10.1 Comparing external transfer rate of disk I/O interfaces



- The external interface capacity usually exceeds the internal or sustained transfer rate of any individual device. Only systems that use multiple disk devices make full use of a high-capacity I/O interface.
- Bus-mastering I/O controllers use less CPU resources. This is particularly important on I/O-intensive server machines. SCSI is generally bus-mastering, and newer PCI EIDE interfaces are bus-mastering. IDE is not.
- Use a disk controller with built in cache memory. The controller cache reduces the need for the operating system to use system RAM for disk cache.
- Don’t assume all disks of a given size perform equally; research performance ratings made by independent testing labs.

Distributing I/O

Disk device I/O is orders of magnitude slower than physical memory accesses or CPU cycles. There is a delay while the disk device seeks the data requested. While an application is waiting for data it has requested from a disk device, it is advantageous for the application to spend the time executing other tasks. One appropriate way to do this is to spread multiple data requests over multiple devices. While one disk is preparing to return data, the application requests another disk to start seeking another set of data. This is called *distributed I/O* or *parallel I/O*.

This section describes ways you can persuade InterBase to distribute I/O over multiple disk devices.

Using RAID

You can achieve up to a ten times performance improvement by using RAID.

RAID (redundant array of inexpensive disks) is a hardware design that is intended to give benefits to performance and reliability by storing data on multiple physical disk devices. It is transparent for software applications to use RAID, because it is implemented in the operating system or at the hardware level. InterBase uses operating system I/O interfaces, so InterBase supports RAID as would any other application software.

Disk striping (included in RAID levels 0, 3, or 5) provides performance benefits by distributing I/O across multiple disks.

Hardware RAID is faster than software RAID or software disk mirroring. RAID implemented with software provides only protection from hard disk failure; it is actually slower than operating without RAID.

Using multiple disks for database files

Similarly to RAID, you can distribute files of a multifile InterBase database among multiple physical disk drives.

For example, if you have a server with four physical disks, **C:**, **D:**, **E:**, and **F:**, and a 10GB database, you can create your database to take advantage of parallel I/O with the following database creation statement:

```
CREATE DATABASE 'C:\data\bigdata1.gdb' PAGE_SIZE 4096
   FILE 'D:\data\bigdata2.gdb' STARTING AT PAGE 1000000
   FILE 'E:\data\bigdata3.gdb' STARTING AT PAGE 2000000
   FILE 'F:\data\bigdata4.gdb' STARTING AT PAGE 3000000;
```

Using multiple disk controllers

If you have so much disk activity on multiple disks that you saturate the I/O bus, you should equip the server with multiple disk controllers, and connect the multiple drivers to the controllers as evenly as possible.

For example, if you have sixteen disk devices hosting database files, you might benefit from using four disk controllers, and attaching four disks to each controller.

Making drives specialized

A database server makes heavy use of both the operating system's virtual memory page file and of temporary disk space. If possible, equip the server with multiple disks and configure the virtual memory file, temporary directory, and database files on separate physical disk devices. This can use parallel I/O to the fullest advantage.

For example, on Windows NT you could locate the operating system files and **pagefile.sys** on **C:**, the temporary directory and infrequently-used files on **D:**, and database files on drives **E:** and higher.

Change the location of the virtual memory file with **Control Panel | System | Performance | Virtual Memory**.

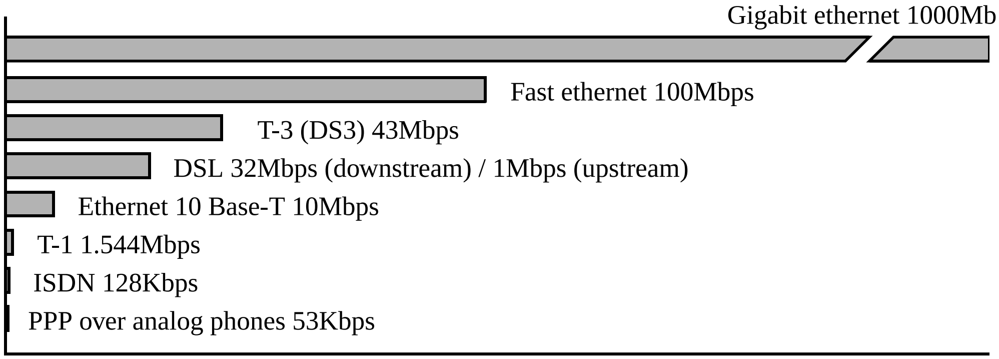
Change the location of the InterBase temporary directory by either specifying a system environment variable INTERBASE_TMP, or editing the **ibconfig** file and specifying the path of the appropriate directory as a value for the TMP_DIRECTORY entry (see **"Configuring sort files" on page 52**).

Using high-bandwidth network systems

For client/server systems, hardware that supports high network bandwidth is as important as I/O capacity. The speed of the network often becomes a bottleneck for performance when many users are making demands on the network simultaneously.

Inexpensive 10 Base-T ethernet equipment is common today, but this technology is bare minimum for LAN configuration. It is recommended to use at least 100 Base-T for a high-performance network. The following graph illustrates relative bandwidth rates for various network interface technology.

FIGURE 10.2 Comparing bandwidth of network interfaces



The maximum bandwidth of gigabit ethernet extends beyond the scale of the graph above.

At the time of this writing, most gigabit ethernet network interface cards (NICs) provide only 600 to 700Mbps bandwidth. Switches, routers, and repeaters also have constrained capacity. It is expected that the state of this technology will continue to improve.

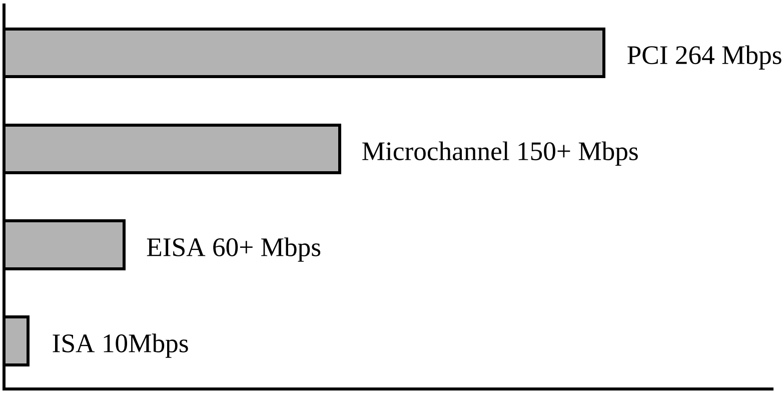
It is recommended that you research reviews and experiment to learn the true throughput of all network hardware in your environment. The slowest component ultimately determines the true throughput.

Tip Network cables develop flaws surprisingly frequently. The result can be sporadic lost packets, for which operating systems compensate by automatically resending packets. This translates into mysterious network performance degradation. You should test network cables regularly. Replacing flawed cables is a low-cost way to keep your network running at peak efficiency.

Using high-performance bus

Bus is important for both I/O controllers and network interface hardware.

FIGURE 10.3 Comparing throughput of bus technologies



While 32-bit full-duplex PCI bus is capable of up to 264Mbps, PCI cards actually range from 40Mbps to 130Mbps.

Tip Use controllers on an integrated local PCI bus, it's faster than peripheral cards that plug into the motherboard.

Useful links

- The T10 Committee home page:
  `http://www.symbios.com/t10/`
  This is a useful place to find information on various storage interface technology.
- PC Guide Hard disk interface & configuration:
  `http://www.pcguide.com/ref/hdd/if/index.htm`
- The SCSI Trade Association:
  `http://www.scsita.org`
  News and vendor information about the state of SCSI technology and products.
- `The Gigabit Ethernet home page:`
  `http://www.gigabit-ethernet.org/`

- The Fibre Channel home page.
  `http://www.fibrechannel.com/`
  Fibre Channel (FC-AL) is an emerging extended bus technology for network, storage, video transmission, and clustering.

Operating system configuration

After you have equipped your server hardware appropriately, you should spend time tuning your operating system for server performance.

Disabling screen savers

Screen savers can have a serious impact on the performance of a server. Because servers are often set aside in a machine room, it's easy for the performance impact of a screen saver to be overlooked. Screen savers demand a surprising amount of CPU resources to run, and these programs run continuously, 24 hours a day.

Screen savers are evasive in their ability to disappear when a database administrator logs in to the console to diagnose a mysterious drop in performance. The server seems responsive to the database administrator as soon as she touches the server, but the speed degrades soon after she leaves the server.

Not all screen savers have the same performance cost. The Windows *OpenGL* screen savers perform continuous floating-point computations to draw three-dimensional shaded shapes in real time. They demand up to 90% of the system CPU, and cause InterBase and other services to slow to one-tenth their normal speed.

The Windows *Marquee* screen saver is one of the least demanding ones, especially when it is configured to pass text across the screen slowly. Some system administrators like to configure a Marquee on each screen in the machine room, to display the respective machine's hostname. This becomes a machine-name label, in raster form.

A screen saver can also be entertainment, but these should be reserved for workstations. A server in a machine room should be unattended, not used as a workstation.

If you must have phosphor burn protection for a monitor that you leave on, get an *Energy Star* approved monitor that has a power conservation mode. This mode blackens the screen after a configurable period of idleness. This not only protects against phosphor burn, but it conserves power. This is like a simple black screen saver, but it is handled by the electronics of the monitor, instead of by software.

The best option is to simply turn off the monitor when you aren't using it. This saves the phosphors, saves electricity, and decreases the amount of heat in the machine room.

Console logins

Don't leave the console logged in on a Windows database server. Even if the desktop is idle, it might use as much as 30% of the machine's CPU resources just maintaining the interface. You should log out of the server's console when you aren't using it. IBConsole enables you to perform most InterBase maintenance and monitoring tasks from another workstation, without logging in at the server's console.

Sizing a temporary directory

When you configure a temporary directory (see **"Managing temporary files" on page 52**), choose a location that has plenty of free disk space. For some operations such as building an index, InterBase can use a great deal of space for sorting. InterBase can even use an amount of space up to twice the size of your database.

The effects of insufficient temporary space include rapid virtual memory page faults, called *thrashing*, which causes a dramatic performance penalty. Another possible effect is a series of "I/O error" related messages printed to the **interbase.log** file on the server.

Use a dedicated server

Using a server for both workgroup file and print services and as a database server is like letting another user play a video game on your workstation. It detracts from the performance of the workstation, and it's not the intended use for the machine.

Use a secondary server as the file and print server, and a new machine for the database server. Alternately, use the secondary server for InterBase, depending on the relative priority of these tasks—the database server benefits from having a dedicated machine, even if it is not the fastest model available. Whatever is the most important service should be given the best machine as dedicated hardware.

If performance is a high priority, you can spend money more effectively by buying a dedicated machine instead of trying to increase resources such as RAM on a machine that is providing another competing service. Compare the cost of the hardware with the cost of having less than maximum performance.

Similarly, it is best to put a database on a dedicated drive, so that the database I/O doesn't compete with the operating system virtual memory paging file or other operating system I/O. See **"Making drives specialized" on page 254**.

Optimizing Windows NT for network applications

It is recommended to set the Windows NT server to optimize for network applications. Without this setting, you might see the CPU usage of InterBase peak for a few seconds every minute. With this setting, these peaks should vanish.

On Windows NT Server, the server is configured by default to give priority to filesharing services. You can change this configuration on the server: **Control Panel | Network | Services | Server**. In the Optimization panel, choose Optimize Throughput For Network Applications.

This change can result in a dramatic improvement of performance for InterBase, as well as other services.

Understanding Windows NT pitfalls

Windows NT has a peculiar way of balancing processes on SMP machines. If a process is exercising one CPU and the other CPU is relatively idle, Windows NT tries to switch the context of the process to the less burdened CPU. On a dedicated database server, the **ibserver** process is likely to be the only significant user of CPU resources. Unfortunately, Windows NT still tries to reassign the context of the process to the other CPU in this case. Once Windows NT has moved the **ibserver** process to the idle CPU, the first CPU becomes less burdened. Windows NT detects this and tries to move **ibserver** back to the first CPU. The second CPU becomes less burdened. This continues many times per minute, and the overhead of switching the process context between the CPUs degrades performance.

There are several possible solutions:

- Run **ibserver** on an SMP server that has enough other duties to occupy the other CPU
- Run **ibserver** only on a single-CPU machine
- Assign CPU affinity to the **ibserver** process:
    1. Launch the Task Manager
    2. Highlight the **ibserver** process
    3. Right-click to raise a window that includes CPU affinity settings

This technique works only if you run **ibserver** as an application, not as a service. If you run InterBase as a service, you must use the Windows API to programmatically set the CPU affinity of the **ibserver** process.

On some operating systems, using a ram disk is a technique for forcing very heavily used files to be in memory, but still allow them to be opened and closed like any other file. If you consider using a ram disk on Windows NT, be aware that the Microsoft ram disk utility for Windows NT uses paged memory to allocate the ram disk. The ram disk itself can be paged out of RAM and stored on the physical disk in **pagefile.sys**. It is futile to use a ram disk on Windows NT to create a high-performance filesystem.

Understanding Linux pitfalls with InterBase Classic

If you are using InterBase Classic on a Linux machine, you may have TCP/IP performance issues. This is due to a TCP/IP kernel driver implementation on Linux called the Nagle algorithm. You can disable this feature on Linux by rebuilding the Linux kernel with the No Nagle option.

IMPORTANT If you are using InterBase SuperServer on Linux, you do not need to disable the Nagle algorithm.

You can also turn off the Nagle algorithm just for the InterBase server. Follow the steps below:

1. Write the C code below into a file called **set_tcp_nodelay.c**.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>

void main(int argc, char *argv[])
{
 int value = 1;

 setsockopt(0, IPPROTO_TCP, TCP_NODELAY, (char *)&value, sizeof(int));
 setsockopt(1, IPPROTO_TCP, TCP_NODELAY, (char *)&value, sizeof(int));
 setsockopt(2, IPPROTO_TCP, TCP_NODELAY, (char *)&value, sizeof(int));

 execl(argv[1], NULL);
}
```

2. Compile this program:

```
gcc -o set_tcp_nodelay set_tcp_nodelay.c
```

3. Move the program to an appropriate location:

```
mv set_tcp_nodelay /usr/local/bin
```

4. Edit your **/etc/inetd.conf** and change the *gds_db* entry. Find the arguments that by default read:

```
/usr/interbase/bin/gds_inet_server gds_inet_server
```

5. Change these arguments to:

```
/usr/local/bin/set_tcp_nodelay gds_inet_server
    /usr/interbase/bin/gds_inet_server
```

6. Force **inetd** to reconfigure itself based on the new **inetd.conf** entry:

```
kill -HUP inet_pid
```

Understanding NetWare pitfalls

NetWare 4.x and 3.x has no technology for supporting virtual memory. All server memory resources rely on physical RAM. The InterBase database server runs as a NetWare Loadable Module (NLM) on the server, and is bound by the server memory configuration. You must equip the NetWare server with enough RAM to operate, under the assumption that there is no virtual memory page file.

InterBase generally requires moderate amounts of memory under most conditions. It should be adequate to equip a NetWare server with 64MB of RAM. Memory requirements can increase under several conditions:

- Database metadata increases in complexity, especially with large numbers of triggers or stored procedures
- The number of simultaneous users increases
- User applications submit complex queries to the server

If InterBase is slow or does not function under these conditions, add RAM to the server until the problem abates.

You can use Novell NetWare as a file and print server in addition to an InterBase database server, but the InterBase server module is given a lower priority than the file/print service. File/print services always have the highest priority, over all other NLMs. The result is that as users read and write files on NetWare volumes belonging to the InterBase server machine, the performance of InterBase (and that of all other NLMs on that server) suffers. The solution is to give InterBase a dedicated server that does not function as a file server. If you cannot dedicate a separate NetWare server for InterBase, expect the performance of NLM services such as InterBase to degrade at times of peak demand on the file and print services.

Network configuration

This section describes performance considerations you should know when configuring a network configuration.

Choosing a network protocol

InterBase supports three protocols: NetBEUI when connecting to a Windows NT/2000 server, IPX/SPX when connecting to a Novell NetWare server, and TCP/IP when connecting to any server. See **"Network protocols" on page 59** for more details.

▸ *NetBEUI and IPX/SPX*

You can use NetBEUI on a network with fewer than 20 users, and IPX/SPX on a network with fewer than 400 users, without significant performance costs. Use TCP/IP if you have more active users on your network simultaneously.

NetBEUI and IPX/SPX are network protocols designed for use on small local area networks. These protocols are commonly used for filesharing services. They are connectionless protocols, which means they broadcast packets to the entire network. This causes a growing amount of "noise" on a LAN. Noise, from the point of view of any given host, can be defined as network traffic that is not intended for the given host. On a LAN with many hosts, enabling NetBEUI or IPX/SPX can overwhelm the network and reduce the available bandwidth for everyone to use. On most enterprise networks, IT experts discourages use of NetBEUI and IPX/SPX.

▸ *TCP/IP*

TCP/IP is a connection-based protocol, which means packets are routed to the intended recipient. This reduces the saturation of the network and the load on individual hosts. There is effectively more bandwidth available to all hosts, and a large number of hosts can share the same network with less performance penalty.

Configuring hostname lookups

Each host on a TCP/IP network has a designated IP address, and TCP/IP traffic is routed to hosts by address. TCP/IP requires a mechanism for clients to translate hostnames to their numeric addresses. Each client host can store the hostname/address associations in a file called **hosts**. You can alternately store this information on a central server, and the clients then retrieve the information on demand using a protocol called DNS. The client requests that the DNS server resolve a hostname, and the server returns the IP address. Then the client can use the IP address to communicate directly with the intended destination. In this configuration, the client must keep only one IP address locally: that of the DNS server host.

Depending on the load on the network and the DNS server itself, hostname resolution can take several seconds. This translates directly into delays when making a network connection. This is related to the message you might see in a web browser, “Looking up host name…” followed by, “Connecting to host name…” This indicates the delay while querying a DNS server to resolve a hostname.

You can speed up hostname resolution. Instead of relying on DNS, add the hostname/address mapping of the database server to the **hosts** file on the client computer. The client can resolve the hostname to its address much faster and more reliably by looking it up in a local file than by querying a service running on another host over the network. This reduces the hostname resolution delay when initiating connections to hosts listed in the local **hosts** file.

Note If you use this technique and later change the address of your database server, you must manually update the hosts files on each client workstation. Depending on the number of workstations in your enterprise, this can be tedious and time consuming. That’s why DNS was invented, to centralize TCP/IP address administration. The suggestion to keep the database server address in a local file is intended to provide improved connection performance, but you should be aware of the administrative workload that it requires.

Tip If you object to the general IP address administration tasks required by using TCP/IP (independently from the DNS issue), consider using DHCP to simplify the task of assigning and tracking IP addresses of each host on the network. InterBase works in a DHCP environment as long as the client host has some means to resolve the server’s IP address correctly at the time a client application requests an InterBase connection.

Database properties

Changing database properties can give an improvement in performance without changing anything in the design of your database. Applications require no change in their coding or design. Property changes are transparent to the client and database design.

Choosing a database page size

InterBase pages are 4KB by default. A typical production InterBase database gains 25 to 30 percent performance benefit by using this page size, relative to smaller page sizes. This page size results in better performance for the following reasons:

- Fewer record fragments are split across pages

  It is common for records to be larger than a single page. This means that InterBase fragments records and stores them on multiple pages. Querying a given record requires multiple page reads from the database.

  By increasing the size of a page, InterBase can reduce the number of multiple page reads and can store record fragments more contiguously.

- Index B-trees are more shallow

  Indexes are B-trees of pointers to data pages containing instances of specific indexed values. If the index B-tree is larger than one page, InterBase allocates additional database pages for the index tree. If the index pages are larger, InterBase needs fewer additional pages to store the pointers. It is easier for the database cache to store the entire B-tree in memory, and indexed lookups are much faster.

- I/O is more contiguous

  It is fairly likely for a query to request successive records in a table. For example, this is done during a table scan, or query that returns or aggregates all records in a table. InterBase stores records on the first page that is unused, rather than ensuring that they are stored near each other in the file. Doing a table scan can potentially require retrieval of data by seeking all over the database. Seeks take time just as reading data takes time.

  Any given page can store records from only one table. This indicates that a larger page is certain to contain more data from the same table, and therefore reading that page returns more relevant data.

- Default number of cache buffers is a larger amount of memory

  InterBase allocates the database cache in number of pages, rather than a fixed number of bytes. Therefore defining a larger page size increases the cache size. A larger cache is more likely to have a better hit rate than a smaller cache.

- Most operating systems perform low-level I/O in 4096 byte blocks

  InterBase performs a page read or write at the OS level by reading in 4096 byte increments regardless of the size of the database page. Therefore, by defining the database with a page size of 4096, the database I/O matches the low-level I/O and this results in greater efficiency when reading and writing pages.

Although 4KB seems to be the best page size for most databases, the optimal size depends on the structure of the specific metadata and the way in which applications access the data. For this reason, you should not consider the 4KB page size guideline to be a magic value. Instead, you should perform testing with your application and database under several different page sizes to analyze which configuration gives the best performance.

Setting the database page fill ratio

Data pages store multiple versions of data records, as applications update data. When a database is restored, the **gbak** utility fills pages with data only up to 80% of the capacity of each page, to leave space for new record version deltas to be stored, hopefully on the same page with the original record. But in a database that is used mostly for reading data rather than updating it, applications never benefit from this 80% fill ratio. In this case, it makes sense to restore data using the full capacity of each page. By storing 25% more data on each page, it reduces the amount of record fragmentation and increases the amount of data returned in each page read. You can specify the option to use all the space of every page for storing data during a database restore using the command:

```
gbak -c -use_all_space backup_file.gbk database_file.gdb
```

Sizing database cache buffers

InterBase maintains a cache in the server's RAM of database pages currently in use. If you have a highly active database, you can gain some performance benefit by raising the default cache from its default of 2048 database pages. As with any cache system, at some point you find diminishing returns. Some experimentation with your particular application and database reveals that point.

See **"Configuring the database cache" on page 124** for details about server cache settings.

The **ibserver** process running on an InterBase server maintains a cache in memory of recently used data and index pages. Like any cache, it depends on repeated use of data on a given page to help speed up subsequent access. In InterBase SuperServer implementations, the cache is shared by all clients connected to the database.

By default, InterBase allocates enough memory for 2048 pages per database. If the page size of the current database is 4KB, then **ibserver** uses 8MB of memory. If the page size is 8KB, then **ibserver** uses 16MB of RAM for cache. The InterBase API provides a method for any individual client to request that the size of the cache be higher. You can set a property on an individual database that establishes a different default cache size when any client connects to that database:

```
gfix -buffers 5000 database.gdb
```

The default of 2048 assumes that the server has a sufficient memory configuration to allocate for 8MB of RAM per database. If memory is less plentiful on your server, or you have many databases that require simultaneous access, you might need to reduce the default number of cache buffers.

It is highly recommended to increase the cache size for a database if you have enough memory to accommodate it. Consider the following points:

- It is not useful to raise the cache size so high that the memory used by **ibserver** starts to page into virtual memory. That defeats the benefit of caching data from disk in memory.
- It is not useful to raise the cache size higher than the number of pages in the database (which you can view with View Database Statistics in IBConsole, or with the **gstat** command-line program). There's no benefit to this, since any given page from disk occupies only one page in the cache, and isn't duplicated.
- One block of memory is allocated for cache per database. If a client connects to two separate databases on one server, the **ibserver** process maintains two separate cache areas of memory. For example, if **database1.gdb** has a default cache size of 8000 pages of 4KB each, and **database2.gdb** has a default cache size of 10,000 pages of 2KB each, then while both databases have at least one connection, **ibserver** allocates a total of 32MB + 20MB of RAM.

You should experiment with larger cache sizes and analyze the performance improvements. At some point, you will observe diminishing returns. A typical application can achieve up to 30% performance increase from proper cache sizing.

The InterBase server does not use more than 512MB of cache per database, so you should not configure the number of cache buffers so high that it exceeds this amount of RAM.

Buffering database writes

InterBase on Windows platforms implements a *write-through* cache by default. Every write operation to a page in cache is immediately written out to the operating system's disk I/O, which itself might have a cache.

By contrast, a *write-back* cache defers flushing of the contents of a given cache page until a later time. InterBase performs multiple writes to a cache page in RAM before it writes the page out to disk. This results in better response time for the majority of write operations. Write-back cache consolidates I/O efficiently, and therefore it is much faster than write-through cache.

InterBase offers write-back cache as the default on UNIX and Linux, and as an option on Windows and NetWare platforms. You can configure this at the database level using **gfix -write async** or by disabling forced writes for the database in IBConsole (**Database Properties | General tab | Options**).

The real benefit of using asynchronous writes (write-back cache) is about four times performance in the typical case. Some users have reported up to 20 times performance improvement from configuring asynchronous writes, in applications that make heavy use of write operations (INSERT, UPDATE, DELETE). The more writing an application does to the database—including write operations spawned by triggers—the more benefit the application gains.

The risk of asynchronous writes is that data in cache might be lost if the server has a power loss, or if **ibserver** exits abnormally for any reason. Write-through cache protects against data loss, at some performance cost. If you test your server host and client/server application thoroughly and they aren’t susceptible to crashes, then it is highly recommended to use asynchronous writes.

Tip Use an uninterruptible power supply (UPS) to help protect your server against sudden power loss. A modest UPS is inexpensive relative to the cost of losing your data, and easy to install. This can allow you to gain the benefits of the asynchronous I/O mode in safety.

Database design principles

This section presents guidelines for database design techniques that benefit performance.

Defining indexes

Proper use of indexes is an important factor in database performance. Effective policies for defining and maintaining indexes can be the key to a very high performance client/server system. The self-tuning nature of indexes in InterBase greatly benefits performance, but you can gain some additional benefit by periodic maintenance tasks.

What is an index?

An index in InterBase is a Balanced-Tree data structure stored inside the database file that provides a quick lookup mechanism for the location of specific values in a table. Queries make use of appropriate indexes automatically by means of the cost-based optimizer, which analyzes the tables and columns used in a given query and chooses indexes that speed up the searching, sorting, or joining operations.

Defining indexes for some columns is part of designing a production database. Indexes dramatically improve performance of SELECT queries. The greater the number of rows in the table, the greater the benefit of using an index. Intelligently analyzing your database and defining indexes appropriately always improves performance.

Indexes incur a small cost to maintain the index B-tree data structure during INSERT and UPDATE operations. Because of this cost, it is not recommended to be overly liberal with index definitions. Don't create redundant indexes, and don't make an index on every column as a substitute for database usage analysis.

You shouldn't define an index for columns that have few distinct data values. For example, a column FISCAL_QUARTER might have only four distinct values over a potentially very large data set. An index doesn't provide much benefit for retrieval of data with this kind of distribution of values, and the work required to maintain the index tree might outweigh the benefits.

What queries use an index?

InterBase uses indexes to speed up data fetching for the following query elements:

- Primary and foreign keys
- Join keys
- Sort keys, including DISTINCT and GROUP BY
- Search criteria (WHERE)

In general, you should define indexes on all columns that you use in JOIN criteria or as sorting keys in an ORDER BY clause. You don't have to define indexes on primary or foreign key columns, because these table constraints implicitly create indexes.

What queries don't use indexes?

InterBase doesn't employ an index in the following operations, even if an index exists for the specified columns:

- Search criteria for CONTAINING, LIKE, and <> inequality operations
- Columns used in aggregate functions, like COUNT()
- Other expressions, like UPPER()

▸ *Directional indexes*

Indexes are defined as either ASCENDING or DESCENDING. To sort in both directions, you need one index of each type. This is also very important if you are using a scrolling list in a Delphi form, or when using the *TTable.Last* method.

Normalizing databases

Design your database with proper normalization of data. Records that have lots of repeating groups of fields are larger than they need to be. Large records can increase the cost of sorting, and also cause records to span more pages than is necessary, resulting in more page fragmentation and needlessly large databases.

Denormalized table design can be more convenient for some types of client applications. You can use InterBase *views* and *stored procedures* to in effect store a denormalized query on the server, for convenient access from client applications. Meanwhile, the physical storage of the data is kept in a more efficient, normalized form.

See the *Data Definition Guide* for details on views and stored procedures.

Choosing Blob segment size

A Blob is a datatype with an unbounded size. It can be many megabytes in size, much larger than any database interface can handle in a single I/O transfer. Therefore, Blobs are defined as a series of segments of uniform size, and the I/O interface transfers Blobs one segment at a time.

Blobs are a special case because there is a special Blob page type, on which other datatypes cannot be stored. The data page for a record containing a Blob stores a Blob ID, which indicates which Blob page the Blob is stored on. A Blob is stored on the same page as the primary record version, if it fits. If it does not fit on that page, special pages are allocated for the Blob--as many as are required--and an index is stored on the primary page. Blob pages are never shared; either a Blob is on a normal data page, or it has a page to itself.

It is advantageous to define a Blob with a segment size equal to the page size. If both the page size and the Blob segment size are 4096 bytes, queries of large Blobs can achieve a data transfer rate of up to 20MB per second. InterBase ceases to be any kind of bottleneck in this situation; it is more likely that the hardware I/O bus, the network bandwidth, or the middleware are the limiting factors for throughput.

Database tuning tasks

This section describes ways you can perform periodic maintenance on your database to keep it running with the best performance.

Tuning indexes

Periodic maintenance of indexes can improve their performance benefit. You can write SQL scripts to automate these tasks. See **"Using SQL scripts" on page 245**.

Rebuilding indexes

Periodically, a B-tree data structure might become imbalanced, or it might have some values in the tree that have been deleted from the database (this should not happen in InterBase versions later than 5, due to index garbage collection).

You should periodically rebuild indexes by turning them off and on:

```
ALTER INDEX name INACTIVE;
ALTER INDEX name ACTIVE;
```

Recalculating index selectivity

The selectivity of an index is an indicator of its uniqueness. The optimizer uses selectivity in its cost-based analysis algorithm when deciding whether to use a given index in a query execution plan. If the selectivity is out of date and doesn't accurately represent the state of the index, the optimizer might use or discount the index inappropriately. This doesn't usually have a great performance penalty unless the selectivity is highly out of date.

You should recalculate the index selectivity if a change to the table affects the average distribution of data values:

```
SET STATISTICS INDEX name;
```

Performing regular backups

There are several performance-related benefits to doing periodic backup and restore of an InterBase database. See **"Benefits of backup and restore" on page 143**.

Increasing backup performance

- Disable garbage collection if you're just going to replace the database immediately anyway; this can make the backup execute faster.

- Back up to a different disk drive.

Increasing restore performance

- Restore from a different disk drive.
- Disable indexes on restore; this makes the restore execute faster so you have a usable database quickly. You must then have to manually activate the indexes after the restore is complete.

Tip Create a SQL script with all the ALTER INDEX statements necessary to activate your indexes, and keep that handy. Use it like a batch file with **isql -i script.sql** to help automate this procedure. You can create this script with this query:

```
SELECT 'ALTER INDEX ' || RDB$INDEX_NAME || ' ACTIVE;'
FROM RDB$INDICES
WHERE RDB$SYSTEM_FLAG = 0 OR RDB$SYSTEM_FLAG IS NULL;
```

You can get the database up and restored more quickly, then activate indexes afterwards. The data is accessible even if the indexes are inactive, but it's slower to query the tables.

Facilitating garbage collection

By default, InterBase databases have a built-in function to automatically sweep old record versions when they become too numerous. However, sweeping is partially inhibited by outstanding active transactions. If the server cannot do complete garbage collection, it has to do extra work to maintain each client's snapshot of the database.

Design your client applications to explicitly start and COMMIT transactions promptly, to reduce the number of outstanding transactions.

See **"Overview of sweeping" on page 121** for more details on sweeping, garbage collection, and the database snapshot.

Application design techniques

This section describes general application programming methods for InterBase, that help to create high-performance clients.

Using transaction isolation modes

InterBase's multigenerational architecture requires that any query or other operation be associated with an active transaction. Without a transaction, an operation has no context with which to maintain its snapshot of the database. IBConsole and BDE tools do a certain amount of automatic transaction management, but it is helpful for performance to manually start and finish transactions.

In the InterBase server engine, a snapshot is generated by making a copy of the state of all other transactions in the database. This snapshot is static for the current transaction. This means that any data committed to the database after the snapshot is created is not visible to operations using that snapshot. This is the repeatable read transaction mode. Two identical queries made at different times are guaranteed to get the same result set, even if other clients are updating data in the database.

Starting a transaction and making a snapshot data structure for the new transaction incurs some amount of overhead. This overhead is magnified when using automatic transaction-handling, because the typical automatic transaction behavior is to start a new transaction and commit it for every statement executed against the database!

Another mode the default mode for BDE is called read committed. In this mode, the snapshot is updated every time the state of any transaction changes. This allows operations in the current transaction to view or act on data that has been committed since the snapshot was created. Updating the snapshot also costs a little bit in performance, so it is recommended to always use the repeatable read mode in InterBase. To do this, configure BDE driver flags to the value 512 or 4608.

Using correlated subqueries

Subqueries are SELECT statements which are included as a clause or expression within another statement. They are typically used to generate a value or result set that are used in conditions of the superior query.

A correlated subquery is one in which the conditions of the subquery are different for each row in the parent query, because they depend on values that vary from row to row. InterBase executes the subquery many times, once for each row in the parent query. Evaluating each row has a large cost in performance relative to a non-correlated subquery. InterBase optimizes non-correlated subqueries out of the loop, executes once, and uses the result as a fixed dataset.

Example as correlated subquery:

```
SELECT * FROM DEPARTMENT D
```

```
WHERE EXISTS (SELECT * FROM EMPLOYEE E
   WHERE E.EMP_NO = D.MNGR_NO AND E.JOB_COUNTRY = 'England')
```

Example as join:

```
SELECT D.*
FROM DEPARTMENT D JOIN EMPLOYEE E
   ON D.MNGR_NO = E.EMP_NO WHERE E.JOB_COUNTRY = 'England'
```

InterBase's optimizer executes a non-correlated subquery once, and uses the result set as many times as necessary in the parent query.

Sometimes a correlated subquery is necessary, given the semantics of the SQL language. However, these types of queries should be used with care and with the understanding that their performance is geometric in relation to the size of the dataset on which they operate.

Preparing parameterized queries

Any dynamic SQL (DSQL) statement must go through a cycle of parse, prepare, and execute. You can submit a DSQL statement to go through this process for each invocation, or you can separate the steps. If you have a situation where you execute the same statement multiple times, or the same form of statement with different parameters, you should explicitly prepare the statement once, then execute it as your looping action.

With *parameterized queries*, you can prepare a statement, but defer supplying the specific values for certain elements of the query.

InterBase supports parameterized queries in DSQL, for cases when a given statement is to be executed multiple times with different values. For example, loading a table with data might require a series of INSERT statements with values for each record inserted. Executing parameterized queries has a direct performance benefit, because the InterBase engine keeps the internal representation and optimization of the query after preparing it once.

Use parameterized DSQL queries in Delphi by following these steps:

1. Place a named parameter in the statement with the Delphi *:PARAMETER* syntax. in place of a constant value in a query. InterBase supports parameters in place constants; tables and column names cannot be parameterized.
2. Prepare the statement. Use the TQuery method *Prepare*. Delphi automatically prepares a query if it is executed without first being prepared. After execution, Delphi unprepares the query. When a query will be executed a number of times, an application should always explicitly prepare the query to avoid multiple and unnecessary prepares and unprepares.

3. Specify parameters. For example, with the TQuery component, use the *ParamByName* method to supply values for each parameter in the query.
4. Execute the statement. SELECT statements should use the *Open* method of TQuery. INSERT, UPDATE, and DELETE statements should use the *ExecSQL* method. These methods prepares the statement in SQL property for execution if it has not already been prepared. To speed performance, an application should ordinarily call Prepare before calling *ExecSQL* for the first time.
5. Repeat steps 3 and 4 as needed.
6. Unprepare the query.

In some real-world cases involving repetitive operations, using parameterized queries has increased performance 100%.

Designing query optimization plans

The *optimization plan* describes the way the optimizer has chosen to execute a query. For certain types of queries, the optimizer might not select the truly optimal plan. A human can analyze different alternate plans and specify a plan overriding the optimizer's analysis. The result can be amazing improvements in performance for some types of queries. In some dramatic cases, this has been used to reduce a 15 minute query to three seconds.

The elements of plan selection are:

- Assigning indexes
- Combining indexes
- Determining join order
- Generating rivers
- Cost estimation
- Sort merges

InterBase supports syntax with the SELECT expression in embedded SQL and DSQL to allow the user to specify the PLAN for a query. The syntax also works with SELECT statements in the body of a view, a stored procedure, or a trigger.

It is beyond the scope of this chapter to describe in detail the syntax of the PLAN clause for specifying the execution plan, or techniques for analyzing queries manually. The section on SELECT in the *Language Reference* includes some examples of using PLAN.

Deferring index updates

Inserting and updating data requires indexes to be updated, which can cause performance to suffer during data INSERT or UPDATE. Some cost incurred while data is entered can result in a big performance win during later data queries.

To minimize the performance hit during INSERT, consider temporarily disabling indexes during high-volume INSERTs. This “turns off” the indexes, making them unavailable to help speed up queries, but also making them not be updated by data INSERTs. Then re-enable the indexes after INSERTing data. This updates and rebalances the indexes once for all the inserted data.

Application development tools

This section describes ways you can develop applications that are efficient, using various popular development environments and tools.

InterBase Express™ (IBX)

InterBase engineers at Borland have created a full-featured set of data-aware VCL components for use with the *TDataSet* architecture in Delphi 5. IBX can also be used with Borland’s C++ Builder. See the *Developer’s Guide* for full documentation of InterBase Express.

IB Objects

Another set of VCL components is available for projects with Delphi. It is designed to provide very sophisticated data component technology that is optimized for use with InterBase. The demo product can be downloaded from `http://www.ibobjects.com`.

Borland Database Engine

You should change the default values for BDE driver options in the BDE Administrator. This section provides guidelines for the driver options, and recommends values that you should use for better performance.

▸ *BDE driver flags*

The recommended value for the DRIVER FLAGS is 4608.

By adding 512 to the DRIVER FLAGS in BDE Config tool, you specify that the default transaction mode is *repeatable read* transactions. This reduces the overhead that automatic transaction control incurs.

By adding 4096 to the DRIVER FLAGS, you specify that the InterBase SQL Links driver should use *soft commits*. Soft commits are a feature of InterBase that let the driver retain the cursor when committing changes. Soft commits improve performance on updates to large sets of data. When using hard commits, the BDE must refetch all the records in a dataset, even for a single record change. This is less expensive when using a desktop database, because the data is transferred in core memory. For a client/server database like InterBase, refreshing a dataset consumes the network bandwidth and degrades performance significantly. With soft commits, the client retains the cursor and doesn't perform a refetch.

Driver flags	Isolation level	Commit type
0	Read committed	Hard commit
512	Repeatable read	Hard commit
4096	Read committed	Soft commit
4608	Repeatable read	Soft commit

TABLE 10.1 Matrix of BDE driver flags values

Caveat: soft commits are never used in explicit transactions started by BDE client applications. This means that if you use explicit transaction start and commit, then the driver flag for soft commit is not used.

SQL passthru mode

The recommended value for this property is SHARED NOAUTOCOMMIT.

SQLPASSTHRU MODE specifies whether the BDE and passthrough SQL statements can share the same database connections. In most cases, SQLPASSTHRU MODE is set by default to SHARED AUTOCOMMIT. If however, you want to pass SQL transaction control statements to your server, you must use the SQL Explorer to set the BDE SQLPASSTHRU MODE to NOT SHARED. Depending on the quantity of data the client handles, you can achieve up to 10 times performance improvement by using the SHARED NOAUTOCOMMIT setting.

Use explicit transaction control and avoid autocommitted statements. Use the following methods: *TDatabase.StartTransaction*, and *TDatabase.Commit*.

SQL query mode

The recommended value for this property is SERVER.

The Active Server of InterBase includes a dynamic SQL parser and execution engine. In order for BDE to execute your SQL queries by sending them to the InterBase SQL engine, you must choose the value SERVER in this property. Otherwise, BDE parses and executes your query, which it does by fashioning a new SQL query and executing it by sending it to the InterBase server. There is no benefit to forcing BDE to reconstruct SQL that you have already written, only performance cost.

Visual components

This section describes visual components that developers commonly use in Delphi and C++Builder to access data from InterBase. Follow the recommendations below for better client/server performance.

Understanding fetch-all operations

In a client/server configuration, a "fetch-all" is the nadir of performance, because it forces BDE to request that the database generate a dataset again and send it over the network.

InterBase and most relational databases do not keep datasets in cache on the server in case the client requests a refresh. InterBase must execute the SQL query again when the BDE requests a refresh. If the query involves a large quantity of data, or complex joining or sorting operations, it is likely to take a long time to generate the dataset.

It is also costly for the server to transfer a large dataset across a network interface. It is more costly by far than it is for a desktop database like Paradox to return a dataset, because a desktop database typically runs locally to the application

It is often the case that software developers choose to use a relational database like InterBase because they are managing a larger amount of data than a desktop database like Paradox can handle efficiently. Naturally, larger datasets take more time to generate and to send over a network.

The person using the client application perceives that it has better performance if the user doesn't have to wait for refreshes. The less often the client application requests a refresh of the dataset, the better it is for the user.

IMPORTANT A principle of client/server application design is therefore to reduce the number of costly refresh operations as much as possible.

TQuery

- *CachedUpdates* = False

  Allows the server to handle updates, deletes, and conflicts.

- *RequestLive* = False

Setting *RequestLive* to False can prevent the VCL from keeping a client-side copy of rows; this has a benefit to performance because it reduces the network bandwidth requirement

- Below are some operations in which a TQuery perform a fetch-all. Avoid these as much as possible, or be aware of the cost of such operations.

Using the *Locate* method

You should use *Locate* only on local datasets.

Using the *RecordCount* property

It's convenient to get the information on how many records are in a dataset, but when using InterBase, calculation of the *RecordCount* itself forces a fetch-all. For this reason, referencing the *RecordCount* property takes as much time as fetching the entire result dataset of the query.

A common use of *RecordCount* is to determine if the result set of an opened TQuery contains any records, or if it contains zero records. If this is the case, you can determine this without performing a fetch-all by testing for both EOF and BOF states. If both end of file and beginning of file are true for the dataset, then no records are in the result set. These operations do not involve a fetch-all.

For example, for a given *TQuery* instance called *qryTest*:

```
qryTest.Open;
if qryTest.BOF and qryTest.EOF then begin
   // There are no result set records.
end
else begin
   // There are some result set records.
end;
```

Using the *Constraints* property

Let the server enforce the constraint.

Using the *Filter* property

For the TQuery to filter records, it must request a much larger dataset than that which it subsequently displays. The InterBase server can perform the filtering in a much more efficient manner before returning the filtered dataset. You should use a WHERE clause in your SQL query. Even if you use a WHERE clause, any use of the *TQuery.Filter* property still forces a fetchall.

▸ *TTable*

The *TTable* component is designed for use on relatively small tables in a local database, accessed in core memory. *TTable* gathers information about the metadata of the table, and tries to maintain a cache of the dataset in memory. *TTable* refreshes its client-side copy of data when you issue the *TTable.post* method and when you use the *TDatabase.rollback* method. This incurs a huge network overhead for client/server databases, which tend to have larger datasets and are accessed over a network. You can observe the activity of *TTable* with the SQL Monitor tool. This reports all calls to the BDE and InterBase API.

Though *TTable* is very convenient for its RAD methods and its abstract data-aware model, you should use it sparingly with InterBase or any other client/server database. *TTable* was not designed to be used for client/server applications.

CHAPTER

11 Data Replication

This chapter documents Synectics Software's InterBase replication server, IBReplicator. It covers the following topics:

- IBReplicator and its components
- Data replication
- How to use IBReplicator

About IBReplicator

IBReplicator is a third-party data replication system created by Synectics Software Ltd. You can use IBReplicator to ensure that changes to the data in any InterBase database can be duplicated in any number of other InterBase databases, even databases with different structures. IBReplicator is included on the "Server" version of the InterBase CD-ROM. If you do not have this available, but are interested in IBReplicator functionality, you can download an evaluation copy of IBReplicator from Synectics Software, at **http://www.synectics.co.za/**.

IBReplicator consists of several executables, in two categories:

- Configuration tools
- Replication server

Configuration tools

Below are the available IBReplicator configuration tools:

Executable	Description
IBRplManager.exe	Replication Manager: a GUI tool that runs only on Windows operating systems, but can be used to configure replication for all platforms.
IBScheduler.exe	Replication Scheduler: a Windows NT/2000 service, used to schedule replication on any platform
IBReplSrvcInstall.exe	A GUI tool, used to install the Windows NT/2000 Service versions of various InterBase/Replication tools

TABLE 11.1 IBReplicator configuration tools

Replication server

On Windows platforms, there are two available executables:

- **IBReplserver.exe** is the GUI version used for testing and replication on Windows 98/ME
- **IBRepl.exe** is the Windows NT/2000 Service version

On UNIX and Linux platforms, there are two available executables:

- **replserver** is the replication server; it runs as a daemon
- **replmgr** is the command-line tool used to start and stop the replication server

InterBase support

The Replication Server can replicate from/to InterBase 6 databases,or from/to InterBase 5 databases, but the Replication Server and the Replication Manager must run on a machine that has an InterBase 6 client installed. The Replication Configuration database must be created on an InterBase 6 server.

Requirements

This section describes supported platforms, supported InterBase versions, and OS requirements.

Windows system requirements

REPLICATION MANAGER, REPLICATION SERVER

Microsoft Windows NT4.0, Windows 2000, Win 98

Memory: 16 megabytes minimum; 64 or more recommended

Processor/Hardware model: 486 minimum; Pentium II recommended

Disk space: 2MB for each application

InterBase Version: 5.*x* and 6.0

Sun Solaris 2.5.x or 2.6.x system requirements

Memory: 32 megabytes minimum; 64 or more recommended

Processor/Hardware model: SPARC or UltraSPARC

Disk space: 2MB for the application

InterBase Version: 5.*x* and 6.0

Linux system requirements

Essentially the same as for Windows NT/2000. The replication server runs on any version of Linux supported by InterBase.

IBReplicator features

This section describes the range of functionality available in IBReplicator:

Overview

- **Fast** Replication occurs directly between servers: there are no intervening layers of processing from database engines or drivers, for example. Our benchmarks indicate that actual replication speeds range from five operations per second over a 28.8 dial-up connection to 200 operations per second between two 200Mhz Pentium machines on a 10BaseT network, where an “operation” is an insert, an update or a delete.
- **Small** IBReplicator consists only of the necessary code; there is none of the overhead associated with any form of middleware. The Replication Engine requires 2MB of disk space on the Server and a further 2MB on the Replication Manager machine. In a Windows environment these may run on the same machine.

InterBase advanced features

- **Advanced datatypes** IBReplicator can replicate all supported InterBase datatypes, including Blobs and Arrays. It handles multi-segment primary keys where each segment can be any supported InterBase datatype.
- **Event alerters** Replication can occur in response to database events.
- **Internationalization** IBReplicator international character sets are supported.

Ease of use

A replication tool is, in its nature, a complex piece of software, especially if it is a highly configurable one, like IBReplicator. Nonetheless, Synectics Software's Replication Server is exceptionally easy to install and use:

- **Point-and-click configuration** This tool allows you to select which tables and fields are to be replicated, and to view and edit optional settings; it also generates the required triggers on the source database for you. A Publish and Subscribe model is used to configure replication schemas.
- **Minimal configuration** Target databases need no configuration at all. The Replication Server can automatically map all tables and fields in the source database to tables and fields of the same name on the target database.
- **Manual Conflict Resolution** This tool allows logged conflicts to be resolved manually.
- **Help** The Replication Manager includes a comprehensive, context-sensitive online help file with detailed explanations of procedures and controls.

Components of IBReplicator

IBReplicator is a system that includes several components, of which the management tool is the most used. The following is an overview of these components.

- **The Replication Server** performs the actual replication of your data. It is available as a service under Windows NT/2000 (**IBRepl.exe**).

  Note On Windows NT/2000, the utility (**IBReplSrvcInstall.exe**) installs, runs, stops, and removes the Replication service. Make sure the correct path to **ibrepl.exe** is displayed when you install the service.

- **The configuration database** saves the details of which data to replicate, and where it should be replicated to (these replication specifications are called *schemata*). You can create as many configuration databases as are required.

- **The Replication Manager** (**IBRplManager.exe**) provides a central user interface from which you can define schemata, manage your configuration databases, and run several supplied utilities. It also includes a comprehensive, context-sensitive online help file with detailed explanations of procedures and controls.
- **The scheduler** defines the intervals at which replication should occur. This tool is available as a service under Windows NT/2000 (**IBScheduler.exe**), and also as an application so that it can be run under Windows 98/ME (**IBReplScheduleManager.exe**). It is run from within the Replication manager.
- **Replication Monitor** This tool provides a real-time graph showing the status of ongoing replications. It also provides information and statistics on all related connections, throughput and activity in the environment. The Monitor can be found in the Tools menu within the Replication Manager.

About data replication

Essentially, a replication server ensures that changes to a *source* database are duplicated on other, *target*, databases so that all databases contain the same data. These source and target databases can be on the same machine, but are usually separated and need to be connected with a local- or wide-area network, or with a dial-up connection.

In principle, this is a straightforward idea, but reality quickly introduces complications. For example:

- **N-way replication** A single database can be *both* a source of data *and* a target of other databases at the same time (see **"Choosing the source database" on page 299**).
- **Complex replication** Source and target databases are often structured differently: tables can have a different number of columns with different names and data-types, some tables can be present in some databases and absent in others, and so on (see **"Choosing replicated tables and stored procedures" on page 302**).
- **Conflicts** Typically, target databases are maintained by their own users, who are adding, deleting and updating rows without knowing about the changes made on a distant source database at the same time. So, a replication server might discover that a row that has been updated is entirely missing on a target, or it might find it needs to add a row to a target already using its primary key (see **"Customizing default settings" on page 297** and **"Choosing the source database" on page 299**).
- **Replication timing** In some contexts, it is enough for replication to happen every so often, perhaps monthly, or daily, or only in response to explicit requests. This *asynchronous* replication requires only a low-end network setup and can be scheduled to occur during idle times such as at night or over weekends, but, of course, the various databases are almost always more or less out of date. In contexts where databases must always be up

to date, *synchronous* replication is needed. In this case, changes are replicated as they occur, with all the consequent pressures on the network. In general, a compromise solution is adequate, with critical data being replicated near-synchronously, and less topical data being replicated asynchronously (see **"Customizing default settings" on page 297** and **"Choosing the source database" on page 299**).

- **Data subsetting** Often, only certain rows in a table should be replicated; in some cases it is necessary to replicate certain rows to one target, and other rows to other databases (see **"Choosing replicated tables and stored procedures" on page 302**).
- **Synchronization** In general, changes in source databases only need to be replicated. You don't need to synchronize. Synchronization is useful for tasks such as filling a new, empty database with all the data already in a source database and for bringing a database restored from an old backup up to date. Note that synchronization is used only under these fairly rare circumstances. If a target database has simply been offline, there is no need to synchronize it as normal replication will catch up with the backlog.

IBReplicator handles all these cases, and it takes pains to make the configuration of a replication environment as straightforward and yet as flexible as possible.

You will appreciate, however, that it is imperative that you *design* your environment carefully. For example, it would be a mistake to replicate data from database A to database B, and then to cause B's changes to be replicated back to A, and this can happen easily in cases where A replicates to B, C, D and E, while B is busy replicating to X, Y and Z, which in their turn are replicating backwards and forwards, and so on.

Note Some authorities discuss "published" databases replicating to "subscribed" databases. They are using a metaphor which sees a database as a sort of journal which provides data to other databases that are on its mailing list. IBReplicator simply sees a database as being either a source of information, or as a target for it (or both, of course).

These two models are not quite identical. The published/subscribed metaphor implies that the published side of the replication needs to be configured to provide the data for all its subscribers. This complicates complex replication situations where different subscribers need different subsets of the available data, and also involves reconfiguring the publisher whenever a subscriber with different needs is added.

IBReplicator's source/target model allows each target to specify whatever data it needs, but it does mean that identical targets need to be configured one by one (see **"Cloning targets" on page 302**).

How IBReplicator works

As you define your replication schemata, the Replication Manager maintains the configuration database in which they are stored. When a schema has been defined, you complete its implementation by telling the Manager to create IBReplicator's *system objects* in the source database.

These system objects are triggers and tables that ensure that every change to the data in the source database is saved in a table called the Replication Log that stores the action that was performed (an insert, update or delete) and the primary key value of the row affected.

The basic sequence

The Replication Server can receive an instruction to replicate from any of several sources:

- Its own internal timer
- Your explicit request
- A timed signal from the **Scheduler**
- An InterBase event

When the server receives such an instruction, it first consults the configuration database to determine which databases are involved. Next it queries its system replication log table in the source database to find the rows that have changed. Finally, it retrieves those rows (which contain the very latest data), and duplicates the action in each target database:

- An *insert* is replicated by inserting a new row with the same primary key values and the same values in each column that is to be replicated.
- An *update* is replicated by finding the row with same primary key and updating its replicated columns to their new values.
- A *delete* is replicated by finding the row with the same primary key and deleting it.

This design helps make IBReplicator easy to use because you need to configure only your source databases; their targets are left untouched.

Furthermore, the design makes the replicator flexible enough for you to organize your network in any way that suits your needs. On the one hand, even the most complex of configurations can be implemented with only a single instance of IBReplicator running on a single server with a single configuration database. But you can also have the replicator running on several or all of your servers, and each instance of the replicator can use its own configuration database (or databases), or various instances can share a configuration, all at your convenience.

IBReplicator's strategies

- **Row-level replication** Also called Data Subsetting. This enables the limiting of data sent to a particular target to a particular domain. For example only replicate rows for Departments 'A' and 'C' to Machine X, and replicate rows for Departments 'S' and 'Q' to Machine Y.
- **Sequence of events** All changes are processed in exactly the same sequence on the target database, as they were originally done on the source database.
- **Sequence of changes** Changes are replicated in their original order.
- **Multisegment primary keys** IBReplicator recognizes primary keys made up of multiple columns, and allows each column to be of any InterBase datatype.
- **Supertransactions** IBReplicator replicates only complete transactions. The Replication Server bundles transactions into supertransactions for efficiency.
- **Encryption** All passwords that are stored in a database or in the system registry are encrypted.
- **Field names** Target tables need not have exactly the same field names and datatypes as their sources.
- **Altered replication schemas** The server can reload configuration settings on command from the Configuration Tool.

Resolving precedence issues

Each Source/Target database pair can have its own conflict resolution settings. IBReplicator provides three ways to handle cases where replicated data conflicts with existing data in the target database:

- Priority-based

  Databases can be given priorities, and the database with the higher priority takes precedence.

  If the source database has precedence the following occurs:

  - An UPDATE finding no identical key record in the target database is converted into an INSERT.
  - An INSERT finding a record in the target database with an identical key is converted into an UPDATE.
  - A DELETE finding no identical key record in the target database is ignored.
- Time-stamped

Older rows in the source database will not overwrite newer rows in the target.

- Master/slave

  The source database always takes precedence.

Replication involving new InterBase 6 datatypes

In InterBase 6, data in NUMERIC and DECIMAL fields that have a precision greater than 9 is stored as INT64. This means that when replicating to InterBase 5, the data may not fit, especially if the InterBase 5 field is being stored as an INT32. The Replication Server checks to see if the data fits, and returns an error if it does not.

InterBase 6 TIME and DATE and TIMESTAMP fields replicate to InterBase 5 DATE fields. IBReplicator adds the current date to the TIME field in order to make it into a proper IB 5 DATE field.

Operation logging

Each Source/Target database pair can have its own error/information logging settings and log file.

IBReplicator can record its transactions in a log that can be either a window on screen or a disk file according to a minimum severity level. You can choose to have it write any of the following items to the error log:

- All errors (for example: can’t connect to target, record already exists, no record to delete, record locked).
- Replication statistics: performance and warnings
- Database connections made
- Values of Key Fields
- SQL Statements generated

Viewing schema

IBReplicator provides a *Schema view* utility that provides a visual representation of your replication setup. It can help you to confirm that your design has been implemented correctly.

To run the Schema View utility, go to the Replication Manager and choose **Tools | Schema view**.

Running the Replication Server

The server portion of IBReplicator can run on Windows, Linux, and Solaris. On Windows it can run as either an application or a service. This section describes the binaries and their functions.

Windows platforms

There are two executables for the various windows platforms **IBReplserver.exe**, which is a standard GUI executable, and IBRepl.exe which runs as a service. The service version can only run on Windows NT or on Windows 2000, while the GUI version can run on any of the windows platforms.

A utility, **IBReplSrvcInstall.exe**, has been provided to install/run/stop/remove the Replication service. Make sure that the correct path to the **Ibrepl.exe** executable is displayed when installing the service.

The Replication Manager contains a Replication Scheduler, which can be configured to invoke the Replication Server at the required intervals. This scheduler can also be run as a service. The required executable is called **IBScheduler.exe** and is installed and removed the same way as the Replication Service.

UNIX and Linux platforms

Two executables are supplied on these platforms:

replserver The binary for the server itself

replmgr The binary for the utility that starts and stops the Replication Server

- To start the **replserver** executable directly, pass the relevant parameters to either **replserver** or **replmgr**.
- To stop the server, you must use **replmgr**.

Replication Server parameters:

```
replserver -u[ser] <username> -p[assword] <password>
    -r[ole] <rolename> -mode <window/daemon> configdbpath
```

Replication Manager parameters:

```
replmgr -u[ser] username -p[assword] password
    -r[ole] <rolename> -a[ction] <start/stop/force>
    -mode <window/daemon> configdbpath
```

Notes:

1. You should not run the Replication Server more than once against the same Configuration database. The results are unpredictable, since the same data may be replicated more than once.
2. It is permissible to run the server more than once on the same machine, as long as each instance of the Replication Server is run against a different configuration database.
3. When stopping the Replication Server with '-action stop', you need to specify the same config database path as when you started the server. This is because replmgr uses InterBase Events to communicate with the running Replication Server, so it must connect to the same database.
4. If you are running multiple instances of the Replication Server on one machine (each connecting to its own **config** database), you will need to run '**replmgr -action stop**' multiple times, once each for each active configuration database.
5. There are four available environment variables:

Variable	**Use**
`ISC_USERNAME`	Contains the required username
`ISC_PASSWORD`	Contains the required password
`ISC_ROLENAME`	Contains the required role name
`IBPATH`	Contains the name of the directory where the server executable is located if it is not in **/usr/interbase/bin**

Using IBReplicator

Once you have installed IBReplicator, you need to define your replication schemata so that the Replication Server can identify your source and target databases, log into them, and replicate only the desired data. You use the management tool, Replication Manager, to design replication schemata.

- The replication engine doesn't have to be running on either the Source or the Target database server for replication to occur; it can be running on any machine on your network or intranet.

- The Replication Server is available for most platforms supported by InterBase, including flavors of UNIX, but the management tool is a Win32 application: it must be run on a Windows machine that has access to the machine running the Replication Server.
- IBReplicator creates a default log file in which all fatal errors, including those that occur during start-up, are logged. This file is stored in InterBase's root directory on the machine where the replication server is running; it is called **Replication.log**. You can specify your own log file when you define your replications later on.
- You can extend the standard replication capabilities by writing additional replication logic in stored procedures.

Using Replication Manager

You use Replication Manager to design replication *schemata*: replication specifications.

Replication Manager's main window provides three important controls, which correspond to the three steps needed to define a replication schema for IBReplicator to implement for you:

1. *Configurations* combo-box, where you choose the configuration database that you need to work on.
2. *Databases* tab, where you identify the databases to be involved in replication.
3. *Replications* tab, where you to define the replication schemata which identify the databases, tables, columns and rows to be replicated.

Each of these steps is discussed in the following sections of this chapter.

IMPORTANT The Replication Manager includes a comprehensive, context-sensitive online help file with detailed explanations of procedures and controls.

Managing configuration databases

When you run Replication Manager for the first time, it notifies you that you must create a configuration database. The Replication Server uses this database to establish what it is supposed to replicate, where it should find the data, and where that data should be sent.

Configuration databases are defined in a special Configuration Databases dialog that you access from **File | Configurations** in the Replication Manager. With this tool, you can:

- Create and drop configuration databases.
- Edit each configuration database's connection parameters.

- Make a configuration database the default.
- Add existing configuration databases to the list of those known to IBReplicator, and remove them from the list without dropping the database.

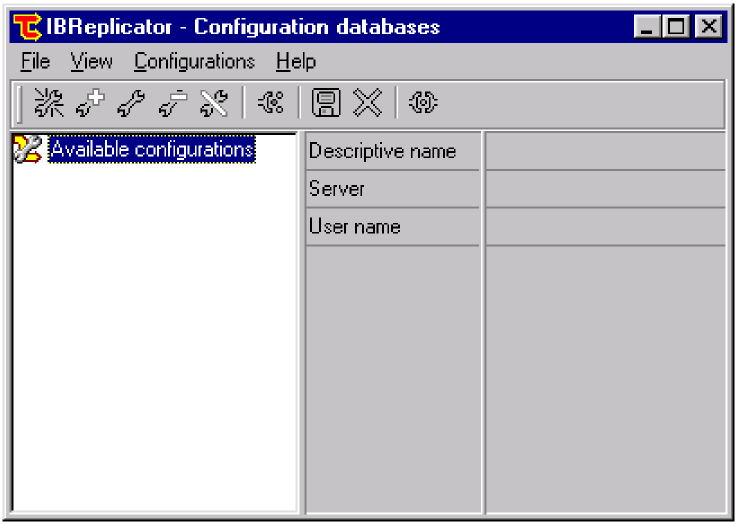
All these functions are available from the toolbar, and from the **Configurations** menu.

Creating a configuration database

To create a new configuration database, follow these steps:

1. In Replication Manager, choose **File|Configurations** to display the Configuration Databases Editor.

FIGURE 11.1 The Configuration Editor



2. In the Configuration Databases Editor, choose **Configurations|Create**.
3. Give the new configuration a descriptive name. This name allows you to identify your configurations when you select the configuration database to edit in the management tool, so the name should be self-explanatory and unique.
4. Supply the path and filename for the new database, and the user name that the Replication Server should use when connecting to it.
5. Choose **Configurations|Save** to create the new database.

Working with configurations

- **To add a configuration database**, choose **Configurations | Add** and follow the same steps used to create a new configuration (see above). The one difference is that the filename you supply *must* identify a configuration database that has already been created.

  Each instance of the Replication Server must have its own configuration database; instances cannot share a configuration.

- **To remove a configuration database**, choose **Configurations | Remove**. Removing a configuration does *not* destroy the database; it simply means that IBReplicator can no longer access it.

- **To destroy a configuration database**, choose **Configurations | Destroy**. This drops the database. Be aware that a dropped database is deleted from the disk!

The default configuration

One of your configuration databases must be identified as the default:

- **To make a configuration the default**, choose **Configurations | Default**. At the prompt, supply the password that the replicator and its tools should use when connecting to that database.

  When you run the Replication Server as a *service* under Windows NT/2000 (see **"About IBReplicator" on page 281**), it always connects to the default configuration. However, when you run the server as an *application* by running **IBReplicator.exe**, you can specify which configuration to use.

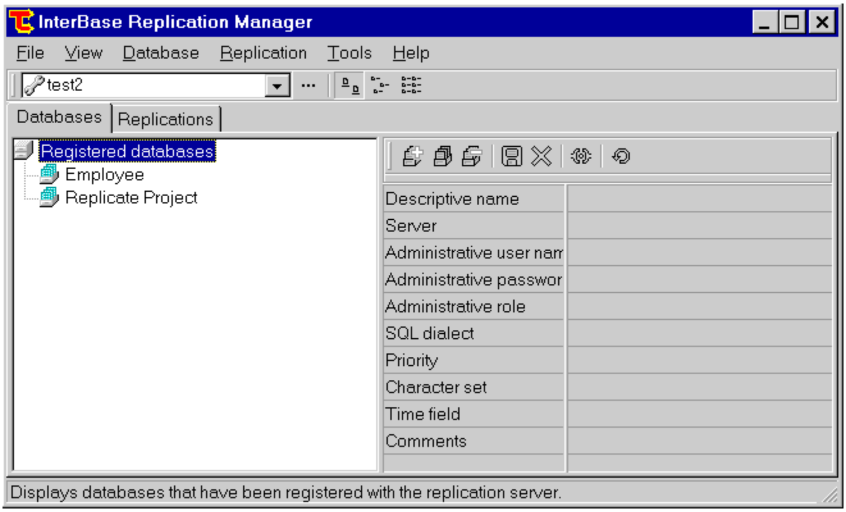
Registering databases

When you have chosen or created a configuration database, you are ready to define the replications that you need, and this is done in Replication Manager's main window. The first step is to identify the databases to be involved using the Databases tab.

You need to register all the databases involved in the replication, both those that serve as sources of data, and those that receive it (the targets). *Registering* a database identifies a database for IBReplicator, and provides values for the parameters it needs to connect to that database.

IMPORTANT Registering a database does *not* create the database, and does not cause any replications to happen.

FIGURE 11.2 Replication Manager database tab



Displays databases that have been registered with the replication server.

The Databases tab provides three controls:

1. A Tree view that lists all the databases that have been registered and saved in the current configuration database.
2. A Field Editor that lists the connection parameters, displays their current values, and allows you to edit those values.
3. A Toolbar that gives quick access to the commands relevant to database registration. These commands are also available from the Databases menu.

Some of the database parameters *must* have values; the others can be set if they are applicable, and ignored if they are not. The essential parameters are:

- *Server* The network path and filename of the database file. Make sure the syntax of this path conforms to the syntax for the kind of network connection that InterBase uses to connect to that database.
- *Administrative Username* and *Password*. The username and password which the management tool should use when connecting to the database. Compare the *replication* username and password used by the Replication Server when replicating (see **"Choosing the source database" on page 299** and **"Choosing the target databases" on page 301**). Of course, the administrative and replication usernames and passwords can be identical.

- The *descriptive name* is another optional but important field. It is optional in the sense that the Replication Server doesn't use it. It is important because it makes your replication schemata easy for users to follow. A name of "Head Office (**employee.gdb**)" is more helpful than the default "New database".

Defining replication schemata

When you have registered the databases that will be involved in replication, your next and final step is to identify the data to be replicated. This is done by defining replication *schemata* on the Replications tab of Replication Manager's main window. This tab contains two controls:

1. A Tree view that displays all the objects to be involved in replications, including source and target databases, tables, and columns.
2. A control panel that presents commands applicable to the selected node in the Tree view. These commands are all available from the Replication menu as well, but only some of the available commands are relevant at any time, and just which these are depends on the node selected in the Tree view. For example, if you have selected a target database, commands for identifying replicated columns are not applicable because you have to identify the table that contains those columns first.

Defining a replication

To specify which data should be replicated and where it should be replicated to, on the Replications tab follow these steps:

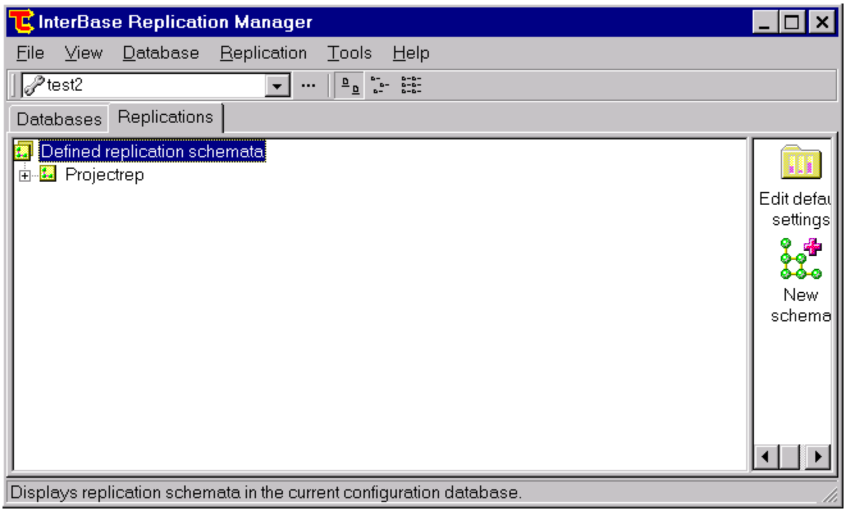
Step	Task	Node	See page
1 (optional)	Customize default settings	Defined Replication Schemata	297
2	Create schema	Defined Replication Schemata	298

Table 1:

Step	Task	Node	See page
3	Choose source database	Source Database	299
4	Choose target database(s)	Target Databases	301
5	Choose replicated tables	Replicated Tables	302
6	Choose replicated columns	Data Columns	308
7	Create system objects	A source database node	309

Table 1:

FIGURE 11.3 Replication Manager Replications tab


InterBase Replication Manager
File View Database Replication Tools Help
test2
Databases
Replications
Defined replication schemata
Projectrep
Edit defau settings
New schema

Displays replication schemata in the current configuration database.

Customizing default settings

Every schema you define has a variety of customizable settings that control replication intervals, conflict resolution strategies, synchronization, and event logging. All of these have been initialized to reasonable values for you, and it is these values that will be supplied to each new schema you create.

To customize IBReplicator's default settings, and to define a new schema, select the root node, Defined Replication Schemata, in the replication Tree view.

To change the default values for these settings, double-click the Edit Default Settings icon in the replication list view or choose **Replication | Default settings** to open the Default Settings dialog.

Setting	Description
Replication timing	Replication can occur at scheduled intervals (defined with the scheduler utility), or in response to an event alert from a source database.
Conflict resolution strategy	When new data from a source database has a primary key that is already used by one of the target databases, IBReplicator can: • Preserve the data from the database with higher priority (as defined when the databases were registered), or • Preserve the most recent version of the data, or • Replicate the source database's version regardless. You can have the Replication Server log conflicts for you to resolve later by choosing **Tools | Conflicts** from Replication Manager's main menu.
Synchronization	*Synchronization* involves updating a target database that is missing data that should have been replicated to it. You can enable synchronization, and you can allow *reverse* synchronization, where the target updates its source.
Event logging	The Replication Server can log its activities for you to inspect, either to a window or to a file.

Table 2:

These defaults are stored in the configuration database, so different configurations can have different defaults. Changing default settings does not change the settings of existing schemata.

Creating schemata

A replication schema contains information about which data should be replicated, where the data can be found, and where it should be sent.

▸ *To define a new schema*

1. Select the replication Tree view's root node.
2. Run the New Schema command by either double-clicking its icon in Replication Manager's control panel, or by choosing **Replication | Schema | New** from the menu. This opens the New Schema dialog.
3. Give your schema a descriptive name.
4. If desired, you can also override the default synchronization settings
5. Choose OK to add a schema node to the Tree view as a child of the Defined Replication Schemata node.

▸ *To edit a schema*

To edit a schema's name and synchronization settings, select a schema node and double-click Edit Schema or choose **Replication | Schema | Edit**.

▸ *To delete a schema*

To delete a schema, double-click Delete Schema or choose **Replication | Schema | Delete**.

Deleting a schema removes *all* the replication information within that schema, and none of that schema's replications will occur from then on.

▸ *The schema node*

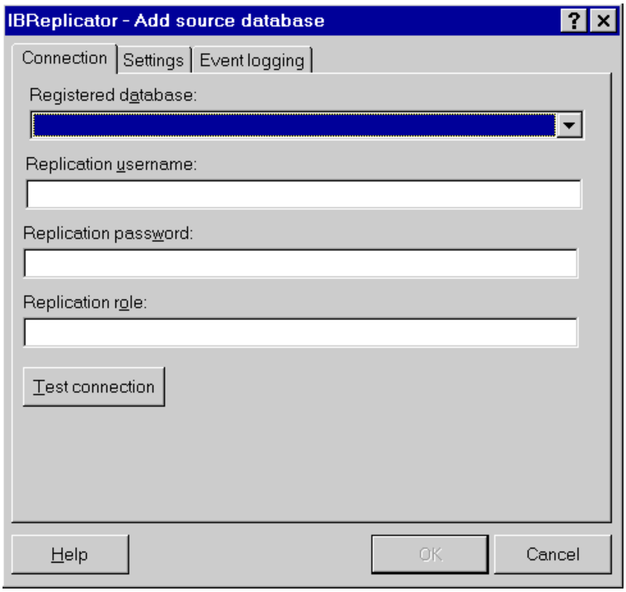
Expanding a schema node reveals two descriptive nodes that exist only to explain their child nodes. The first node contains the source database, and the second contains the target databases.

Choosing the source database

To specify the database from which replicated data will originate, follow these steps:

1. Expand a schema node and select its Source Database node.
2. Double-click the Add Source Database icon or choose **Replication | Source | Add** to open the Source Database dialog.

FIGURE 11.4 Source Database dialog



3. Choose one of your registered databases to be the source of the data that is to be replicated (see **“Registering databases” on page 294**).

4. Supply a username and password and/or a rolename for the Replication Server to use when it connects to that database to retrieve the data to be replicated.

   There are also miscellaneous settings for customizing IBReplicator’s behavior. These are initialized to default values (see **“Customizing default settings” on page 297**). You can override them for the particular source database that you are defining.

5. Click OK to add a source database node to the Tree view as a child of the Source Databases node.

Selecting a source database node gives access to the commands for editing and removing a source database; these commands can also be accessed by choosing **Replication | Source | Edit** or **Replication | Source | Delete**.

There is also a command for creating system objects. This vitally important command is discussed in **“Creating system objects” on page 309**).

- Each schema can have only one source database, so define a schema for each of your source databases.

- A source database in one schema can be a target database too, but only in another schema.

Choosing the target databases

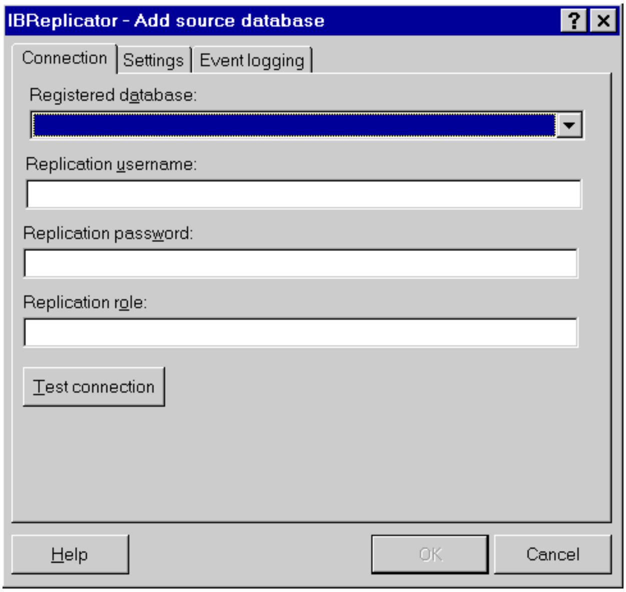
A target database is one to which replicated data can be sent. When identical data is to be sent to more than one target, it is possible to define one target and then copy or "clone" that definition and apply it to other targets. This section describes these operations.

Specifying a target database

To specify a database to which replicated data can be sent, follow these steps:

1. Expand a schema node and select its Target Databases node.
2. Double-click the Add Target Database icon or choose **Replication | Targets | Add** to display the Target Database dialog.

FIGURE 11.5 Add Target Database dialog



3. Choose one of your registered databases as the source of the data to be replicated (see **"Registering databases" on page 294**).

4. Supply a username and password for the Replication Server to use when connecting to that database to retrieve the data which is to be replicated. You may want to specify a replication role as well. (Compare the *administrative* username, password and role described in **"Registering databases" on page 294**).

   You can also use the Target Database dialog to specify synchronization settings that override the schema's settings for that particular target (see **"Creating schemata" on page 298**).

5. Click OK to add a target database node to the Tree view as a child of the Target Databases node.

Editing and deleting targets Selecting a particular target database node gives access to the commands for editing and removing that target database; these commands can also be accessed by choosing **Replication | Target | Edit** or **Replication | Target | Delete**.

▸ *Cloning targets*

The tables, columns and rows to be replicated are defined for each target database involved. This allows different data to be replicated to different targets. Typically, however, all the targets in a schema will be receiving similar or identical data, and setting up numerous identical targets can become very tedious indeed.

You can clone targets within a schema. To clone a target, follow these steps:

1. Either drag a target database node onto the schema's Target Databases node, or select a target database in the Tree view and choose **Replication | Target | Clone**.
2. Supply the name of the target database. All information for its replicated tables, columns, and rows is automatically duplicated.

The new target must have the same structure as the target being cloned.

Note A schema can supply one or several targets for its source database, and different data can be replicated to different targets (see **"Choosing replicated tables and stored procedures" on page 302** and **"Choosing replicated columns" on page 308** for the details).

Choosing replicated tables and stored procedures

The next step is to identify which tables contain data to be replicated.

▸ *Specifying replicated tables*

To identify the tables that contain data to be replicated, follow these steps:

1. Expand a node representing a target database and select the Replicated Tables node.
2. Double-click the Replicated Tables icon in Replication Manager's control panel, or choose **Replication | Tables | Define**. This opens the Replicated Tables and Procedures dialog.
3. Map tables in the target database to the tables in the source database by clicking and dragging them.

The above steps can be done automatically by choosing **Auto Generate Tables, Keys and Fields**. In this case, all tables in the Source database are automatically mapped to tables of the same name in the target database. The Manager does not check that the tables exist in the target database, but assumes that source and target databases are identical.

Tables that have already been mapped are ignored by this process.

In cases where source and target databases are not identical, it is often still appropriate to use one of these options:

- Map all tables, and manually remove/modify different tables

or

- First manually map all different tables, then let the automatic process take care of the rest

Removing replicated tables

Use the Replicated Tables dialog to remove source-target table mappings, or choose **Replication | Tables | Remove**.

Note You are free to define complex mappings where source and target tables have different names and even different columns, provided that:

· The two tables both have primary keys that uniquely identify the rows containing data to be replicated.

· The columns containing data to be replicated have compatible datatypes.

Table settings and row-level replication.

To override a source-target table mapping's default settings (see **"Customizing default settings" on page 297**), follow these steps:

1. Select the node in the Tree view that represents the mapping,
2. Double-click the Table Settings icon in Replication Manager's control panel or choose **Replication | Tables | Settings** to open the Table Settings dialog.

Using WHERE *conditions to identify rows*

You can use the Table Settings dialog to provide a WHERE condition that identifies which rows in the table should be replicated.

The basic syntax of the SQL SELECT statement can be represented informally as:

```
SELECT columnlist
FROM   table
WHERE  condition
```

As an example, a request to find all unpaid invoices might look like this:

```
SELECT *
FROM   SALES
WHERE  PAID = "n"
```

So, if you want to replicate only rows detailing unpaid invoices, set the Row-level Replication Condition in the Table Settings dialog to:

```
:PAID = "n"
```

This is a very simple condition, but there is no limit to the condition's complexity: any condition that is valid in a SELECT statement is valid as a replication condition.

IMPORTANT T*est the* condition with a select statement in IBConsole and then copy and paste it into the replication condition's edit box. A syntactically invalid expression is inconvenient rather than disastrous: your log file will be filled with SQL errors. But a valid condition which identifies the wrong rows can indeed be disastrous: consider the case where the condition is being used to replicate public data and leave confidential data behind. Be careful here!

Note Identify each column name in the replication condition with a prefixed colon; this allows IBReplicator to add the `NEW.` and `OLD.` that the system's triggers need to implement row-level replication.

Replicating to stored procedures

Displaying target stored procedures The Replicated Tables dialog shows the tables in the target database by default. To see the target's stored procedures instead, check the Stored Procedures radio button. This allows you to map *target* stored procedures to the *source* tables.

Sophisticated users can define stored procedures to do the work of inserting, updating, or deleting rows on the target database. This allows you to customize IBReplicator to handle specialized cases.

To define a stored procedure for IBReplicator to call, do the following:

- Provide one parameter for each field to be replicated, with data fields listed first, in alphabetical order, and then the key fields, also in alphabetical order. The last parameter defines the action to be taken. It should be a single character, which the server will set when it calls the procedure. The possible values are:

  `D` The row identified by the key fields' values should be deleted

  `I` A row should be inserted with the supplied values in its fields

  `U` The row identified by the key fields' values should be updated to the supplied values

- *Return* an integer value indicating the action's result:

  `0` Success

  `1`
  - For inserts, a row with the specified key already exists (a primary key violation occurred)
  - For updates and deletes, no row with the specified key was found

  `2`
  - For updates and deletes, too many rows were found to act upon
  - For inserts, an unexpected error that was not a primary key violation occurred.

  `-`
  - The procedure can also raise an exception for the Replication Server to trap. When an exception is raised, the server rolls back its current transaction and the exception appears in the log, unless it indicates a primary key violation that occurred while inserting; in this case, the conflict resolution rules are used to handle the exception.

  In many cases, your procedure will be able to handle both conditions 1 and 2 and will therefore always return zero. For example, an inserted row may be there already, so the procedure can choose to change the insert into an update, or to insert the row somewhere else, or to give it a new primary key. In such cases, the procedure can safely return zero, indicating success.

Example The source database contains this table:

```
CREATE TABLE T (
  K1 INTEGER NOT NULL,
  K2 VARCHAR(10) NOT NULL,
  K3 DATE NOT NULL,
  F1 VARCHAR(20),
  F2 DOUBLE PRECISION,
  F3 INTEGER,
  F4 VARCHAR(100),
  F5 NUMERIC(4,1),
  F6 NUMERIC(9,2),
  F7 NUMERIC(15,2),
  PRIMARY KEY (K1,K2,K3)
);
```

A stored procedure can then be defined on a target database with the same table as follows:

```
CREATE PROCEDURE REPLICATE_T (
  F1      VARCHAR(20),
  F2      DOUBLE PRECISION,
  F3      INTEGER,
  F4      VARCHAR(100),
  F5      NUMERIC(4, 1),
  F6      NUMERIC(9, 2),
  F7      NUMERIC(15, 2),
  K1      INTEGER,
  K2      VARCHAR(10),
  K3      DATE,
  TYPE    CHAR(1)
) RETURNS (RESULT INTEGER)
AS
  DECLARE VARIABLE COUNTER INTEGER;
BEGIN

  RESULT = 0;  /*default return value*/

  SELECT COUNT(*) FROM T
    WHERE K1 = :K1 AND K2 = :K2 AND K3 = :K3
      INTO :COUNTER;

/* Inserts: If the row already exists, then exit with result=1. This
 * causes the Replication Server to apply conflict rules, which will
 * probably cause the procedure to be called again, but with an
 * with a TYPE of “U”. An alternative approach would simply change
 * TYPE to “U” and proceed to update the row instead.
 */
  IF (TYPE = 'I' AND COUNTER > 0) THEN
  BEGIN
    RESULT = 1;
    EXIT;
  END

/* Updates: If the row does not exist, then exit with result=1. This
```

```
 * causes the Replication Server to apply conflict rules, which will
 * probably cause the procedure to be called again, but with a TYPE
 * of “I”. An alternative approach would simply change TYPE to “I”
 * and proceed to insert the row instead.
 */
  IF (TYPE = 'U' AND COUNTER = 0) THEN
  BEGIN
    RESULT = 1;
    EXIT;
  END

/* Deletes: If the row does not exist then exit with result=1. The
 * Replication Server will log the error, but otherwise ignore it.
 */
  IF (TYPE = 'D' AND COUNTER = 0) THEN
  BEGIN
    RESULT = 1;
    EXIT;
  END

  IF (TYPE = 'I') THEN
    INSERT INTO T(K1,K2,K3,F1,F2,F3,F4,F5,F6,F7)
    VALUES (:K1,:K2,:K3,:F1,:F2,:F3,:F4,:F5,:F6,:F7);

  IF (TYPE = 'U') THEN
    UPDATE T
    SET    F1 = :F1,
           F2 = :F2,
           F3 = :F3,
           F4 = :F4,
           F5 = :F5,
           F6 = :F6,
           F7 = :F7
    WHERE  K1 = :K1
      AND  K2 = :K2
      AND  K3 = :K3;

  IF (TYPE = 'D') THEN
    DELETE FROM T
    WHERE K1 = :K1
      AND K2 = :K2
      AND K3 = :K3;
  EXIT;
END
```

Note InterBase stored procedures do not support Blob and array types as parameters, so such columns cannot be replicated with stored procedures.

Choosing replicated columns

IBReplicator needs to know which of the columns in each replicated table contain the data that should be replicated, and it also needs to know which columns in the tables should be treated as primary keys.

▸ *Defining primary keys*

1. To identify a replicated table's primary key columns, expand a node representing a source and target table mapping and select the Key Columns node.
2. Double-click the Define Primary Key icon or choose **Replication|Columns|Key** to open the Key Columns dialog
3. To map target and source columns, click and drag the target table's key columns to their corresponding columns in the source table.

▸ *Identifying data columns*

1. To identify the columns that contain data which is to be replicated, expand one of the nodes representing a source and target table mapping and select the Data Columns node.
2. Double-click the Define Data Columns icon or choose **Replication|Columns|Data** to open the Data Columns dialog.
3. To map columns in the target table to their corresponding columns in the source table, click and drag them across.

Creating system objects

When you have completed a replication schema by specifying a source database and its targets, and mapping the source and target tables and columns to each other, your final step is to tell Replication Manager to create for you the system's tables and triggers in the source database. Select a source database node in the replications Tree view and then double-click the Create System Objects icon in Replication Manager's control panel, or choose **Replication | Source | Create system objects**.

Special user REPL

In a replication source database, any changes made to the database by the user REPL will *not* be replicated. This is useful when you need to make bulk changes to a database, and do not want those changes replicated, as in the following situations:

- Bi-directional replication
- n-way replication

Below are explanations of these special replication situations.

Bi-Directional replication

Bi-Directional replication is used when there are two databases, and each needs to send all changes to the other; they are both source databases, and they are both target databases. The concept is simple, but care needs to be taken to prevent changes from bouncing back and forth in a loop.

In a replication environment, any change to data in a replicated table gets moved from the source database to the target. In a bi-directional setup, this causes problems, because a change made to one database gets sent to the second database, and because it induces a change to the second database, the change gets sent straight back again, in a loop.

Fortunately, there is an easy way to get around this by using the special case user called REPL. This user is not created automatically, but must be created by the DBA, and sufficient privileges must be granted.

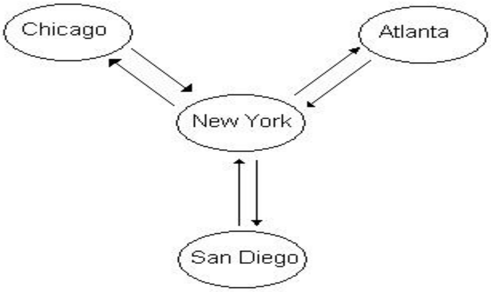
Any change made to a database by the user REPL will not get replicated to the target database.

When setting up each *target* database in a bi-directional environment, tell the Replication Server to log in as the user REPL.

n-Way replication

n-Way replication is replication between more than two databases. This can be set up in a number of ways; in its simplest form, each database replicates from/to all others. In this case it is very similar to Bi-Directional Replication and can be set up in exactly the same way.

In more complex scenarios, more work needs to be done. For example, in Hub-and-Spoke setup, a central database controls the replication, fetching data from each database, and forwarding the data to all of the others:



In this example, the New York database fetches data from Chicago, and then forwards it on to Atlanta and San Diego. Clearly, the data from Chicago cannot be written to the New York database using the Special User REPL, otherwise it would stop there, and not be forwarded to Atlanta and San Diego. But, if the user REPL is not used, then the data will bounce straight back to Chicago.

In order to get around this, create a few new usernames and use the 'Row Level Replication Condition' to prevent data from bouncing back to the originating database.

In the case of Chicago to New York, the Replication Server logs in as REPL_CHICAGO. In the reverse case (New York to Chicago), we log in as REPL, and have a Row Level Condition of ' USER <> REPL_CHICAGO'. This means that data will move from Chicago to New York, and from there onwards to Altanta and San Diego, but will not bounce back to Chicago. Other changes in New York will move to Chicago (amongst others) and stop there because of the use of the user REPL.

Here is a table of the UserNames and conditions required in the above scenario:

Source	Target	UserName	Condition
Chicago	New York	REPL_CHICAGO	
San Diego	New York	REPL_SANDIEGO	
Atlanta	New York	REPL_ATLANTA	
New York	Chicago	REPL	USER < 'REPL_CHICAGO'
New York	San Diego	REPL	USER < 'REPL_SANDIEGO'
New York	Atlanta	REPL	USER < 'REPL_ATLANTA'

This ensures that data is replicated from one branch into the hub in New York. The hub then replicates this data to the other branches, but not back to the originating branch.

Problems and workarounds

The following is a list of problems that some users have encountered along with the techniques for surmounting them.

Problem: Array fields

Array fields are supported, but they are copied across in one unit. This may cause problems for very large arrays.

Workaround

Don’t replicate large array fields.

Problem: Computed fields

If you select a computed field for replication, IBReplicator display an error message and does not replicate *any* data for the table in question. This will be fixed in a later release.

Workaround

Don’t select computed fields for replication.

Problem: Redundant replication

When only certain of a table's fields have been selected for replication, then data in that table is be replicated, even if the changes affect only fields *not* selected for replication.

Workaround

None needed.

Problem: Floating-point primary keys

Fields that store floating-point numbers should *not* be identified as primary key fields in the replication configuration. The inevitable rounding errors make these fields unsuitable for uniquely identifying the rows in a table.

Workaround

This is less of a problem with InterBase 6, where all DECIMAL and NUMERIC data is stored in INTEGER fields. If key fields with a type of FLOAT or DOUBLE PRECISION are used, then the problem will persist. The solution is not to use FLOAT or DOUBLE PRECISION fields as part of the primary key.

Problem: Shared resource error

"Cannot create shared resource" error when trying to run either the replication or the scheduler managers.

Workaround

Shut down the scheduler, and restart it after using the manager.

APPENDIX

A

InterBase Document Conventions

This appendix covers the following topics:

- The InterBase 6 documentation set
- The printing conventions used to display information in text
- The printing conventions used to display information in syntax, code, and examples

The InterBase documentation set

The InterBase documentation set is an integrated package designed for all levels of users. It consists of six full-length printed books plus the *Release Notes and Getting Started, which describes installation and database migration*. Each of these books is also provided in Adobe Acrobat PDF format and is accessible on line. If Adobe Acrobat is not already installed on your system, you can find it on the InterBase distribution CD-ROM or at **http//www.adobe.com/products/acrobat**. Acrobat is available for Windows platforms and most flavors of UNIX.

Book	Description
Operations Guide	Provides an introduction to InterBase and an explanation of tools and procedures for performing administrative tasks on databases and database servers; also includes full reference on InterBase utilities, including **isql**, **gbak**, **gfix**, and others
Data Definition Guide	Explains how to create, alter, and delete database objects using the SQL language
Developer's Guide	Provides both reference and task-oriented material for users of the Borland RAD tools (Delphi, C++ Builder, and JBuilder); includes chapters on writing UDFs, driver configuration, developing embedded installation applications, and using the new InterBase Data Access Components
Language Reference	Describes the SQL language syntax and usage; includes references for procedure and trigger language, InterBase keywords, functions in the InterBase UDF library, error codes, character sets, and the system tables
Embedded SQL Guide	(formerly called the *Programmer's Guide*) Describes how to write embedded SQL database applications in a host language, precompiled through **gpre**
API Guide	Explains how to write database applications using the InterBase API

TABLE A.1 Books in the InterBase 6 documentation set

Printing conventions

The InterBase documentation set uses various typographic conventions to identify objects and syntactic elements.

The following table lists typographic conventions used in text, and provides examples of their use:

Convention	Purpose	Example
UPPERCASE	SQL keywords, SQL functions, and names of all database objects such as tables, columns, indexes, and stored procedures	• the SELECT statement retrieves data from the CITY column in the CITIES table • can be used in CHAR, VARCHAR, and BLOB text columns • the CAST() function
italic	New terms, emphasized words, all elements from host languages, and all user-supplied items	• *isc_decode_date()* • the host variable, *segment_length* • contains six variables, or *data members*
bold	File names, menu picks, and all commands that are entered at a system prompt, including their switches, arguments, and parameters	• **gbak**, **isql**, **gsec**. **gfix** • specify the **gpre -sqlda old** switch • a script, **ib_udf.sql**, in the **examples** subdirectory • the **employee.gdb** database; the **employee** database • the **Session | Advanced Settings** command

TABLE A.2 Text conventions

Syntax conventions

The following table lists the conventions used in syntax statements and sample code, and provides examples of their use:

Convention	Purpose	Example
`UPPERCASE`	Keywords that must be typed exactly as they appear when used	• `SET TERM !!;` • `ADD [CONSTRAINT] CHECK`
italic	User-supplied parameters that cannot be broken into smaller units	• `CREATE TRIGGER` *name* `FOR` *table*`;` • `ALTER EXCEPTION` *name* `'`*message*`'`

<*italic*>	Parameters in angle brackets can be broken into smaller syntactic units; the expansion syntax for these parameters follows the current syntax statement	`WHILE (<condition>) DO`
		`<compound_statement>`

TABLE A.3 Syntax conventions

Convention	Purpose	Example
[ ]	Optional syntax: you do not need to include anything that is enclosed in square brackets; when elements within these brackets are separated by the pipe symbol (|), you can choose only one	• CREATE [UNIQUE] [ASCENDING | DESCENDING] • [FILTER [FROM *subtype*] TO *subtype*]
{ }	You must include one and only one of the enclosed options, which are separated by the pipe symbol (|)	{INTO | USING}
|	You can choose only one of a group whose elements are separated by this pipe symbol	SELECT [DISTINCT | ALL]
...	You can repeat the clause enclosed in brackets with the “…” symbol as many times as necessary	(<col> [,<col>…])

TABLE A.3 Syntax conventions (*continued*)

APPENDIX B

InterBase Limits

This appendix defines the limits of a number of InterBase characteristics. The values the following table lists are design limits, and in most cases are further restricted by finite resource restrictions in the operating system or computer hardware.

Characteristic	Value
Maximum number of clients connected to one server	There is no single number for the maximum number of clients the InterBase server can serve—it depends on a combination of factors including capability of the operating system, limitations of the hardware, and the demands that each client puts on the server. Assuming a “normal” type of client application that executes database operations from human interaction, and a modern server platform (Pentium 150MHz+, 64MB RAM), expect the InterBase server to comfortably handle up to 150 clients. This is a guideline, not a guarantee. Applications that engage in high levels of contention or that perform complex or high-volume operations could cause the practical number of clients to be fewer. Note also that some operating systems do not have the technology to serve 150 incoming network connections.
Maximum database size	The maximum addressable file size for a single file is 2GB on Windows 98, 4GB on most other platforms. Refer to your operating system documentation to verify file size limits. Combined with the multifile database feature of InterBase, this allows many terabytes of addressable file space.
Maximum number of files per database	By design, 2^{16} (65,536), because the files are enumerated with an unsigned 16-bit integer. Shadow files count toward this limit. This is a design parameter of InterBase, but most operating systems have a much lower limit on the number of files that a single process can have open simultaneously. In some cases, the OS provides a means to raise this limit. Refer to your OS documentation for the default open files limit, and the means to raise this limit.
Maximum number of cache pages per database	65,536; for the sake of performance, a more practical upper limit would be 10,000. Total size of cache pages should never exceed 50% of memory.
Maximum number of databases open in one transaction	No restriction. The parameters in a transaction parameter buffer comprise a linked list, so there is no limit except that imposed by system resources.
Maximum number of tables per database	By design, 2^{16} (65,536), because tables are enumerated with a 16-bit unsigned integer.

TABLE B.1 InterBase specifications

Characteristic	Value
Maximum versions per table	255; then no more metadata changes until the database has been backed up and restored.
Maximum row size	64KB. Each Blob and array contributes eight bytes to this limit in the form of their Blob handle. Systems tables (tables maintained by the InterBase engine for system data) have a row size limit of 128KB.
Maximum number of rows and columns per table	By design, 2^{32} rows, because rows are enumerated with a 32-bit unsigned integer per table. Number of columns in a row depends on datatypes used. One row can be 64K. For example, you can define 16,384 columns of type INTEGER (four bytes each) in one table.
Maximum number of indexes per table	By design, 2^{16} (65,536), because indexes per table are enumerated with a 16-bit unsigned integer.
Maximum number of indexes per database	By design, 2^{32}, because you can create 2^{16} tables per database, and each table can have 2^{16} indexes.
Maximum index key size	Starts at 256 bytes for a single-column key, and 200 for multicolumn keys; subtract four bytes for each additional column. Example: a single-column CHAR key can be up to 256 – 4 = 252 bytes; a three-column key must add up to 200 – 12 = 188 bytes. Note that multibyte character sets must fit within the key by counting bytes, not by counting characters. For example, a single-column key using 3-byte UNICODE_FSS characters can have a maximum of (256 – 4) / 3 = 84 characters.
Maximum number of events per stored procedure	No restriction by design, but there is a practical limit, given that there is a limit on the length of code in a stored procedure or trigger (see below).
Maximum stored procedure or trigger code size	48KB of BLR, the bytecode language compiled from stored procedure or trigger language.

TABLE B.1 InterBase specifications

Characteristic	Value
Maximum Blob size	The size of the largest single Blob depends on the database page size: 1KB page size: largest Blob is 64MB 2KB page size: largest Blob is 512MB 4KB page size: largest Blob is 4GB 8KB page size: largest Blob is 32GB A Blob is a stream of many segments. The maximum Blob segment size is 64KB.
Maximum tables in a JOIN	No restriction by design, but the task of joining tables is exponential in relation to number of tables in the join. The largest practical number of tables in a JOIN is about 16, but experiment with your application and a realistic volume of data to find the most complex join that has acceptable performance.
Maximum levels of nested queries	There is no restriction by design. The practical limit depends on the type of queries you nest. Experiment with your application and a realistic volume of data to find the deepest nested query that has acceptable performance.
Maximum number of columns per one composite index	16.
Levels of nesting per stored procedure or trigger	750 on Windows platforms. 1000 for UNIX platforms.
Maximum size of key in SORT clause	32KB.
Range of date values	January 1, 100 a.d. to February 29, 32768 a.d.

TABLE B.1 InterBase specifications

Index