Installing and Running on Unix

Version 4.0

Borland InterBase Workgroup Server

Borland International, Inc., 100 Borland Way
P.O. Box 660001, Scotts Valley, CA 95067-0001

Printed in the U.S.A.

1E0R795
9596979899-9 8 7 6 5 4 3 2 1
C1

Table of Contents

Preface 1

Chapter 1: What is InterBase for Unix? 7

Chapter 2: Unix Installation and Licensing 9

Before You Install 9
- Preparing Your System 9
- Upgrading From a Previous InterBase Version 10
  - Backing Up Previous Version Databases 11
  - Changing Lock Header files and Semaphore Values 11
  - Saving Previous Version Files 11

Installing InterBase on Unix 12

Licensing InterBase 13
- Upgrading From a Previous Version of InterBase 13
- First Time Installation of InterBase. 13
  - iscinstall Messages 16

After You Install. 17
- General Information for All Systems 17
- Information for System Upgrades . . . 18

Testing the Installation 18

Chapter 3: Migrating to InterBase 4.0 19

Migration Topics 19

System Configuration Changes 19
- Unix System Settings 19
- Linking 20
- Library Calls 20

InterBase System Changes 20
- Keywords 20
- Error Codes 21
- gpre 21

New Features for Version 4.0 21
- SQL Descriptors. 21
- gbak Switches 21
- gbak, gfix, and gsec Switches 22
- Shadowing. 22

Version 3.x GDML Features Unsupported in Version 4.0 22
- GDML Features 22
- SQL Select Statement 22
- Access Privileges 23

Components Not Supported in Version 4.0 23
- qli 23
- pyxis and fred 23
- Journaling 23

User Response 23

Chapter 4: Using InterBase on Unix 25

Database Administration 25

Working With Databases 25
- Forced and Buffered Writes 26
- Shutdown and Logoff 26

Building Applications 26
- Using the gpre Precompiler with C++ 27
- Creating User-defined Functions and BLOB Filters 27

Chapter 5: Using Command-line isql 29

Invoking isql 29
- Command-line Options 30
  - Examples 31
- Exiting isql. 31
- Connecting to a Database 32

Transaction Behavior in isql 32

Extracting Metadata 33
- Order of Extraction 34

isql Commands. 35
- SHOW Commands 35
- SET Commands. 37

Other isql Commands 37
BLOBDUMP. 38
Comments 38
EDIT 39
SHELL 39
INPUT 39
Exiting isql. 40
Error Handling 40

Chapter 6: Using the Command-line DBA Utilities 41

Using the Security Utility 41
The Security Database. 41
Security Utility Commands. 42
Displaying the Security Database 42
Adding Entries to the Security Database 43
Modifying the Security Database 44
Deleting Entries from the Security Database 44
Using the Security Utility from the Unix Command Line. 44
Using the Database Maintenance Utility 45
Sweeping a Database 45
Overview of Sweeping 45
Options for Database Sweeping. . . 46
Changing the Sweep Interval 46
Disabling Automatic Sweeping . . . 47
Sweeping a Database at a Specified Time 47
Repairing a Database 47
Options for Database Repair 48
Validating a Database 48
Mending a Corrupt Database 49
Handling Checksum Errors 51
Recovering Limbo Transactions 51
The Two-phase Commit Process 51
Automated Transaction Recovery 52
Options for Transaction Recovery 52
Shutting Down a Database 53
Shutdown Options 54
Preventing Further Attachments 54
Preventing Further Transactions 55
Forcing Shutdown. 56
Restarting a Database 56
Controlling Performance of Forced Writes. 57
Using the Backup and Restore Utilities 57
General Syntax 58
Performing a Simple Backup 58
Backing Up Remote Databases. . . . 59
Backing Up to a Storage Device . . . 59
Performing a Simple Restoration 60
Restoring from a Storage Device 60
Monitoring the Status of Backup and Restore 61
Making a Transportable Backup 61
Backup Options. 62
Backing Up Metadata 62
Preventing Garbage Collection . . . 63
Ignoring Checksums 63
Ignoring Limbo Transactions 63
Restoration Options 64
Splitting a Database into Several Files 64
Making Indexes Inactive 65
Disabling Validity Checking 65
Changing the Database Page Size 66
Restoring a Database Without Its Shadow 67
Restoring Data Incrementally 67

Upgrading to a New On-disk Structure . . . 67

Chapter 7: Shadowing a Database . . . 69

Overview of Shadowing . . . 69
Tasks for Shadowing . . . 69
Advantages of Shadowing . . . 70
Limitations of Shadowing . . . 70
Creating a Shadow . . . 71
Creating a Single-file Shadow . . . 72
Creating a Multi-file Shadow . . . 72
Creating a Shadow in Auto Mode or Manual Mode . . . 73
Auto Mode . . . 73
Manual Mode . . . 74
Creating a Conditional Shadow . . . 74
Activating a Shadow . . . 74
Dropping a Shadow . . . 75
Adding a Shadow File . . . 75

Appendix A: Error Messages . . . 77

Error Messages for Backup and Restore . . . 77
Error Messages for Database Maintenance . . . 81
Error Messages for Security Utility . . . 82

Appendix B: isql Command Reference . . . 83

BLOBDUMP . . . 83
EDIT . . . 84
EXIT . . . 85
HELP . . . 85
INPUT . . . 86
OUTPUT . . . 87
QUIT . . . 88
SET . . . 88
SET AUTODDL . . . 89
SET BLOBDISPLAY . . . 90
SET COUNT . . . 92
SET ECHO . . . 93
SET LIST . . . 94
SET NAMES . . . 95
SET PLAN . . . 95
SET STATS . . . 96
SET TERM . . . 98
SET TIME . . . 99
SHELL . . . 99
SHOW CHECK . . . 100
SHOW DATABASE . . . 101
SHOW DOMAINS . . . 101
SHOW EXCEPTIONS . . . 102
SHOW FILTERS . . . 103
SHOW FUNCTIONS . . . 103
SHOW GENERATORS . . . 104
SHOW GRANT . . . 105
SHOW INDEX . . . 105
SHOW PROCEDURES . . . 106
SHOW SYSTEM . . . 107
SHOW TABLES . . . 108
SHOW TRIGGERS . . . 109
SHOW VERSION . . . 110
SHOW VIEWS . . . 110

Appendix C: Command-line DBA Utilities Reference . . . 111

gbak . . . 111
gfix . . . 114
gsec . . . 116

Index . . . 119

List of Tables

Table 1: InterBase Core Documentation 1
Table 2: Text Conventions 2
Table 3: Syntax Conventions 2
Table 1-1: InterBase Features 7
Table 2-1: License Options. 14
Table 2-2: iscinstall Messages 16
Table 5-1: isql Command-line Options 30
Table 5-2: Order of Extraction. 34
Table 5-3: isql SHOW Commands 36
Table 5-4: isql SET Commands 37
Table 5-5: isql Miscellaneous Commands 37
Table 5-6: SQLCODE and Message Summary 40
Table 6-1: Format of the isc4.gdb Security Database 42
Table 6-2: Summary of gsec Commands 42
Table 6-3: gsec Options 43
Table 6-4: Sweeping Options. 46
Table 6-5: Database Repair Options 48
Table 6-6: Transaction Recovery Options 52
Table 6-7: Shutdown Options 54
Table 6-8: Backup Options 64
Table 7-1: Auto vs. Manual Shadows. 73
Table A-1: Error Messages for Backup and Restore. 77
Table A-2: Messages for Database Maintenance 81
Table A-3: Error Messages for Security Utility 82
Table B-1: isql Commands 83
Table B-2: SET Commands 89
Table C-1: gbak Argument Descriptions. 111
Table C-2: gbak Backup Options 112
Table C-3: gbak Restore Options 112
Table C-4: gfix Options 114
Table C-5: gsec Options. 116
Table C-6: gsec Argument Descriptions. 117

Preface

This preface describes the documentation set, the printing conventions used to display information in text and in code examples, and the conventions a user should employ when specifying database objects and files by name in applications.

The InterBase Documentation Set

The InterBase documentation set is an integrated package designed for all levels of users. The InterBase server documentation consists of a five-book core set and a platform-specific installation guide.

The InterBase core documentation set consists of the following books:

Table 1: InterBase Core Documentation

Book	Description
Getting Started	Provides a basic introduction to InterBase and roadmap for using the documentation and a tutorial for learning basic SQL through **isql**. Introduces more advanced topics such as creating stored procedures and triggers.
Data Definition Guide	Explains how to create, alter, and delete database objects through **isql**.
Language Reference	Describes SQL and DSQL syntax and usage.
Programmer's Guide	Describes how to write embedded SQL and DSQL database applications in a host language, precompiled through **gpre**.
API Guide	Explains how to write database applications using the InterBase API.
Installing and Running on . . .	Platform-specific information on installing and running InterBase.
InterBase Windows Client User's Guide	Installing and using the InterBase PC client. Using Windows ISQL and the InterBase Server Manager.

Printing Conventions

The InterBase documentation set uses different fonts to distinguish various kinds of text and syntax.

Text Conventions

The following table describes font conventions used in text, and provides examples of their use:

Table 2: Text Conventions

Convention	Purpose	Example
UPPERCASE	SQL keywords, names of all database objects such as tables, columns, indexes, stored procedures, and SQL functions.	The following SELECT statement retrieves data from the CITY column in the CITIES table.
italic	Introduces new terms, and emphasizes words. Also used for file names and host-language variables.	The *isc4.gdb* security database is *not* accessible without a valid *username* and *password*.
bold	Utility names, user-defined and host-language function names. Function names are always followed by parentheses to distinguish them from utility names.	To back up and restore a database, use **gbak** or the server manager. The **datediff()** function can be used to calculate the number of days between two dates.

Syntax Conventions

The following table describes the conventions used in syntax statements and sample code, and offers examples of their use:

Table 3: Syntax Conventions

Convention	Purpose	Example
UPPERCASE	Keywords that must be typed exactly as they appear when used.	SET TERM !!;

Table 3: Syntax Conventions (Continued)

Convention	Purpose	Example
italic	Parameters that *cannot* be broken into smaller units. For example, a table name cannot be subdivided.	CREATE TABLE *name* (<*col*> [**,** <*col*> ...]);
<*italic*>	Parameters in angle brackets that *can* be broken into smaller syntactic units. For example, column definitions (<*col*>) can be subdivided into a name, data type and constraint definition.	CREATE TABLE *name* (<*col*> [**,** <*col*> ...])**;** <*col*> = *name* <*datatype*> [CONSTRAINT *name* <*type*>]
[ ]	Square brackets enclose optional syntax.	<*col*> [**,** <*col*> ...]
...	Closely spaced ellipses indicate that a clause within brackets can be repeated as many times as necessary.	(<*col*> [**,** <*col*> ...]);
|	The pipe symbol indicates that either of two syntax clauses that it separates may be used, but not both. Inside curly braces, the pipe symbol separates multiple choices, one of which *must* be used.	SET TRANSACTION {SNAPSHOT [TABLE STABILITY] | READ COMMITTED};
{ }	Curly braces indicate that one of the enclosed options *must* be included in actual statement use.	SET TRANSACTION {SNAPSHOT [TABLE STABILITY] | READ COMMITTED};

Database Object-naming Conventions

InterBase database objects, such as tables, views, and column names, appear in text and code in uppercase in the InterBase documentation set because this is the way such information is stored in a database's system tables.

When an applications programmer or end user creates a database object or refers to it by name, case is unimportant. The following limitations on naming database objects must be observed:

- Start each name with an alphabetic character (A-Z or a-z).

- Restrict object names to 31 characters, including dollar signs ($), underscores (_), 0 to 9, A to Z, and a to z. Some objects, such as constraint names, are restricted to 27 bytes in length.
- Keep object names unique. In all cases, objects of the same type, for example, tables and views, *must* be unique. In most cases, object names must also be unique within the database.

For more information about naming database objects with CREATE or DECLARE statements, see the *Language Reference*.

File-naming Conventions

InterBase is available on a wide variety of platforms. In most cases users in a heterogenous 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 example 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, file specification is denoted as follows:

```
"<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 are secondary. Secondary file specifications generally cannot include a node name, but may specify a full path name. 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]
   [<secondary_file>]

<secondary_file> = FILE "<filespec>" [<fileinfo>] [<secondary_file>]

<fileinfo> = LENGTH [=] int [PAGE[S]] | STARTING [AT [PAGE]] int
   [<fileinfo>]
```

In the secondary file specification, <*filespec*> does not support specification of a node name.

CHAPTER 1

What is InterBase for Unix?

InterBase for Unix is an integrated software package containing both a server and a client.

InterBase servers on Unix work with InterBase clients on Unix and on all other platforms and have the following features:

Table 1-1: InterBase Features

Feature	Description
SQL Enhancements	Declarative referential integrity, triggers, stored procedures, BLOB data types, arrays of data types, updatable views, user-defined functions, and BLOB filters.
Command-line **isql**	Command-line version of InterBase Interactive SQL Tool.
Command-line DBA Utilities	Command-line version of InterBase database administration tools.
gpre pre-processor	Preprocessor for embedded SQL programs.
Header files	Files included at the beginning of application programs that define InterBase data types and function calls.
Example programs	C programs, ready to compile and link, which may be used to query the standard InterBase example database on the server.
Message file	*interbase.msg*, containing messages presented to the user.

Databases on Unix systems can be accessed through an InterBase client of a PC running Windows. For complete information, see the *Windows Client User's Guide*.

CHAPTER 2

Unix Installation and Licensing

This chapter provides general information about installing and licensing InterBase on Unix platforms. For detailed information on configuration requirements on your Unix platform, refer to the Borland World Wide Web site at: HTTP://WWW.BORLAND.COM/. If you are unable to access this information, call InterBase customer support.

Before You Install

This section describes what to do *before* installing InterBase on your system. InterBase requires certain minimum Unix options and configurations, described below in “Preparing Your System.” In addition, if you are upgrading to InterBase Version 4.0 from an earlier version, see “Upgrading From a Previous Version of InterBase,” in this chapter. For platform-specific system requirements, refer to the Borland WWW page.

Preparing Your System

Before you install InterBase, you should verify that the following options exist and are properly configured on your system:

- Kernel facility requirements:
  - IPCSHMEM, the shared memory facility
  - IPCSEMAPHORE, the semaphore facility

  To add these options to the kernel, edit the kernel configuration file and rebuild the kernel. See your Unix system administrator for information.
- SEM_UNDO structures

  On platforms that require SEM_UNDO structures, you must verify that you have approximately one per process running or expect to run on a single machine.

- Free semaphore requirements:

  To determine the total number of semaphores available in your kernel consult your system administrator.

  To determine the number of semaphores currently in use, use the following command:

  ```
  % ipcs -a
  ```

  The total number of semaphores in use is the total of the NSEMS column. InterBase uses three sets of semaphore clusters. If you do not see these sets of semaphores listed in NSEMS or you have other semaphores installed, you may not have enough semaphores to run InterBase. You can drop some of the existing semaphores using:

  ```
  % ipcrm -s <semaphore ID from ipcs display>
  ```

  If the maximum number of semaphores for your system has not already been reached, you can also edit the kernel configuration file to specify more semaphores. However, you then must rebuild the kernel before installing InterBase.

- Kilobyte requirements:

  Verify that you have the required number of kilobytes on the partition where you want to install the InterBase directory tree.

Caution The InterBase directory tree should be located on a disk local to the server on which you are installing InterBase. If the directory is located on a disk of a remote machine, you may not be able to use the InterBase help and example databases.

- */tmp* directory

Important InterBase places files in the */tmp* directory during data processing (e.g., queries, building indexes, sorting data). If you plan on issuing queries that retrieve a large number of records or if you will be using indexes or sorting on large relations, you must increase the amount of temporary space available to InterBase. You can change the temporary working location from */tmp* to another directory by setting the environment variable, TMP, to point to the desired directory.

Upgrading From a Previous InterBase Version

If you have a previous InterBase version installed, you must back up databases created with the previous version, save certain files, and reconfigure your system.

Backing Up Previous Version Databases

To access databases created using a previous 4.0 version, you must back them up before installing this new 4.0 version. The on-disk structure (ODS) 8.0 has been updated to a new version in InterBase Version 4.0. To access databases created using earlier versions of InterBase, back them up using that version of **gbak** *before you install Version 4.0*. After you install the product, restore the databases using the Version 4.0 **gbak**. For information on upgrading the on disk structure, see "Upgrading to a New On-disk Structure" in Chapter 6: "Using the Command-line DBA Utilities."

Changing Lock Header files and Semaphore Values

Before installing InterBase 4.0 on your system, you must change or reconfigure certain files.

Use the **#gds_drop -a** command to remove the associated lock manager and semaphores *before* installing a new version of InterBase. Use the **#ipcs -a** command to verify that the semaphores have been removed.

Note InterBase 4.0 does not use *lock_header.*

The platform-specific installation instructions contain more information for changing configuration files, lock header files, and semaphore values.

Saving Previous Version Files

If you have a previous InterBase version installed, you should save certain files before installing a new version:

- */usr/interbase/isc_license.dat*
- */usr/interbase/isc_config*
- */usr/interbase/isc.gdb*

  If you intent to use this file with InterBase V4.0 databases, you must back it up using the previous version **gbak** and restore it using the V4.0 **gbak** to *isc4.gdb.*

Note If you used Pictor in previous InterBase versions and intend to use it in version 4.0, you will need to save the Pictor files. However, Pictor is unsupported in version 4.0.

Installing InterBase on Unix

The following example is ageneric Unix installation. Since most platforms vary slightly, you will need the platform-specific installation instructions provided on to the Borland WWW page.

1. Log in as superuser or root and insert the distribution medium into the drive.
2. To install InterBase in */usr/interbase*, change your directory as follows:

   ```
   % cd /usr
   ```

   You can install InterBase in any other directory, but after the installation is complete, define the link */usr/interbase* which points to the installed product files.
3. Use the **tar** utility to read the files from the media. The tape contains the */interbase* directory, which has the files and subdirectories that are used during the installation procedure.

   If you are not using the default device, use the **f** option of **tar** and specify the name of the device you are using. The following example uses the default device:

   ```
   % tar -xv
   ```
4. When the **tar** finishes restoring the files, run the installation script:

   ```
   % sh install
   ```

   The installation script moves files from the tape to directories in */inter-base*. It creates links to this directory from */usr/lib* and */usr/include.*

Note If you do not have enough semaphores configured on your machine, you will see this message:

```
% semget failed
```

For information about resetting the number of semaphores, see “Before You Install,” in this chapter. If you do not reinstall InterBase files after resetting the semaphores, you must manually install the sample databases and the help data-base using **gbak**:

5. When your system prompt is displayed, the installation is complete.
6. If you have a network connection and want to use the network immedi-ately, kill the *inetd* process and restart it.
7. Logout as superuser or root.

Licensing InterBase

Upgrading From a Previous Version of InterBase

If you upgraded from a previous version of InterBase, then the license ID you were given for the previous version is still valid. Copy your backed up version of *isc_license.dat* to */usr/interbase/isc_license.dat*. InterBase is now licensed for use.

First Time Installation of InterBase

You must license your copy of InterBase before you can use it. *Licensing is performed directly through your sales engineer.* When you are ready to license InterBase, call your sales engineer and follow the instructions below.

To license your installation:

1. Log in as superuser or root.
2. Run the **iscinstall** utility from the client that you want to license. You will need to have a license file on each client:

```
% /usr/interbase/bin/iscinstall
```

Note To cancel the **iscinstall** utility, type **exit** or **quit** at any of the prompts.

3. Answer each prompt with the information provided by your sales engineer. In some cases, you will use default information and press **Return** to use the default value. The default value for each **iscinstall** prompt is in parentheses. The prompts are described below:

```
Do you need instructions (N)?
```

Type **Y** if you want instructions for using the **iscinstall** utility. If you exit before completing the instructions by typing **exit** or **quit**, you must restart **iscinstall**. If you complete the instructions, you remain within the utility and the next prompt is displayed.

```
License file (/usr/interbase/isc_license.dat):
```

Press **Return** to use the default path name (*/usr/interbase/isc_license.dat*), or type the file name you want to use to store the licensing information. (If you do not use the default file, you must append the licensing information to the *isc_license.dat* file after the licensing procedure is complete.)

```
Check for duplicate licenses (N)?
```

Press **Return** if you do not want to check for duplicates, which may result in duplicate licenses being registered.

Type **Y** to check for duplicates, which will assure that no duplicate licenses are registered.

```
Enter PRODUCT:
```

Type **interbase**

Note If you are licensing the international InterBase version, type **international**.

```
Enter VERSION:
```

Press **Return** unless otherwise instructed by your sales engineer.

```
Enter OPTIONS:
```

Type the options that apply to your license, as instructed by your sales engineer. Do not type commas or spaces between each option.

Table 2-1: License Options

Character	License Option
I	Internal relation access
E	External relation access
R	Remote interface
D	Data definition utilities
S	Remote server
Q	qli
L	**gpre** for all languages, except ADA
A	**gpre** for ADA
3	**gpre** for C++
C	Chinese
K	Korean
N	Japanese

```
Enter ETHER:
Enter NODE:
```

Accept the ethernet ID or node ID depending on your platform requirements. The default ID for the node you are licensing appears during the procedure. Press **Return** to accept the ID. For remote installations, you will need to determine the ID of the node before you run the installation script. Ask your system administrator how to determine a node ID.

```
Enter UNTIL:
```

Press **Return** unless you purchased a temporary license for product evaluation. If you have a temporary license, type the expiration date, using the format, DD-MMM-YYYY.

```
Enter COMMENT:
```

Type an optional comment (80 character limit), or press **Return** if you do not want to enter a comment.

```
Enter ID:
```

Type the license ID provided by your sales engineer.

```
Enter KEY:
```

Type the license key provided by your sales engineer.

4. When you finish entering the licensing information, **iscinstall** displays the information you entered. For example:

```
Entered: PRODUCT interbase, NODE 1700u98a, ID ISC-76, KEY 93-1-2-2
```

   - If the record is not valid, **iscinstall** prints an error message and does not create the license file entry. Verify that the information you entered matches the above instructions and information provided by InterBase.
   - If the record is valid, it is placed in the *isc_license.dat* license file on your system. If you specified a different authorization file in which to place the license record, you must append the licensing information from that file to *isc_license.dat*.

5. **iscinstall** then begins another licensing routine and prompts you for the next InterBase product:

```
Enter PRODUCT:
```

If you are not installing another InterBase product, type **quit** or **exit**.

6. To exit from the **iscinstall** utility, type **N** at the following prompts:

```
License another node (Y)?: n
Another license file (Y)?: n
```

7. Verify that the *isc_license.dat* file has “read” permission for all InterBase users.
8. Log out as superuser or root.

iscinstall Messages

The following table lists **iscinstall** messages and the reasons for their occurrences. Where appropriate, corrective actions are suggested.

Table 2-2: **iscinstall** Messages

Message	Probable Cause
Cannot open help file.	The help file either does not exist or does not have read permission.
Duplicate licenses will be registered.	You typed N in response to the "Check for duplicate licenses" prompt.
Duplicate licenses will NOT be registered.	You typed Y in response to the "Check for duplicate licenses" prompt.
File *<filename>* cannot be read or created. Check the file's permissions.	The license file you want to use cannot be opened. (The most likely cause is that the file does not have read/write permission or you do not have write access to the directory.) After you correct the file permissions, rerun **iscinstall.**
File *<filename>* does not seem to exist. Create it (Y)?	The file in which license information will be stored does not exist. If you type Y, the file will be created.
File *<filename>* doesn't appear to be an InterBase License File... Continue (N)?	The file you specified to use as the license file already exists but cannot be used as an InterBase Authorization File. Type Y to continue and enter a different file name.
ID is mandatory, please re-enter.	You did not enter a required license ID at the "Enter ID" prompt.
Invalid character in key.	You entered one or more incorrect characters in the key value. Re-enter the key, correcting the invalid character(s) shown in the message.
Invalid character in node ID.	You entered one or more incorrect characters in the node value. Re-enter the node, correcting the invalid character(s) shown in the message.
Invalid date.	You entered an incorrect date or did not enter the date in a valid format. Use the date format, DD-MMM-YYYY. For example, January 21, 1991 should be entered as 21-JAN-1991.
Invalid format for key.	You entered the key value using an incorrect format. Re-enter the key using the format shown in the message.

Table 2-2: **iscinstall** Messages (Continued)

Message	Probable Cause
Invalid key value, check entries for accuracy. Please retry.	You entered an incorrect key value. Rerun **iscinstall** and enter the correct key value. If you see this message again, contact InterBase Customer Support.
Invalid license option. Valid options are: *<options list>*	You entered a letter that does not correspond to an option. Re-enter using one or more of the options listed in the message.
Invalid PRODUCT ID.	You entered an incorrect product ID. Re-enter the product ID using one of the product IDs listed in the message.
Key is mandatory. Please re-enter.	You did not enter a key value. Enter the key value provided by InterBase Customer Support.
Node is limited to a maximum of 8 characters. Please re-enter.	You entered a node ID that is longer than 8 characters. Re-enter the correct node ID.
Product is mandatory! Valid entries are: *<entry list>*	You did not enter a product ID. Enter a product ID using one of the entries from the list in the message.
Re-try licensing node (Y)?	You exited **iscinstall** before completing the license by typing exit or quit. To rerun **iscinstall**, press **Return**. If you do not want to install InterBase at this time, type N.

After You Install

After you install and license InterBase, you may need to make additional modifications to your system to use InterBase most effectively. You should also test your installation to make sure InterBase runs correctly. This section describes general changes you might need to make. For platform-specific information, refer to the Borland WWW site. For any late-breaking information, you should refer to the *readme* file located on the product.

General Information for All Systems

- All users who intend to use InterBase must add */usr/interbase/bin* to their paths.
- Any user who accesses a database must have read / write access to that database. To lessen the security risk, you can associate a security class

with the database. For information about security schemes, see the *Data Definition Guide*.

- InterBase 4.0 uses a file called *isc.config* to configure the lock manager. The lock table is managed either by Unix System V shared memory or by mapped fields which can be dynamically expanded as needed. Depending on the operating system and the applications you run, *isc_config* may need to be edited to increase maximum shared memory used for the lock table and the number of semaphores used. See the Borland WWW site for information about configuring your system.

Information for System Upgrades

- Restore any previous version databases using the version 4.0 **gbak.** You must have saved them using the previous version **gbak** before you installed.

Testing the Installation

1. Make sure you added */usr/interbase/bin* to your path.
2. Start command-line **isql**, the interactive data definition and manipulation facility and open the sample database, *employee.gdb*. For example:

```
%isql

isql> connect /usr/interbase/examples/v4/employee.gdb;

isql> show tables;
   . . .
isql> select * from country;
   . . .
isql> quit;
```

3. For more information about using **isql** commands, see Chapter 5: “Using Command-line isql.” For information on executing SQL statements, see the *Language Reference*.

Note Users may elect to use Windows **isql** from a PC. For complete information about using Windows **isql**, see the *Windows Client User’s Guide*. For a complete SQL tutorial using Windows **isql**, see *Getting Started*.

CHAPTER 3

Migrating to InterBase 4.0

This chapter lists considerations for customers of earlier versions of InterBase who are moving to InterBase Version 4.0. New users and users who are not migrating existing applications programs to the new version will not require this information.

There are several differences in the ways that Version 3.3 and Version 4.0 manage data. Application programs that accessed InterBase Version 3.*x* databases will require some modifications to access InterBase Version 4.0 databases.

Important Back up Version 3.*x* databases using the Version 3.*x* **gbak** utility and restore them using the Version 4.0 **gbak** utility.

Backup files made by Version 3.*x* **gbak** should use the **-**t option so that the backup file is transportable to other platforms. The Version 4.0 **gbak** creates transportable backup files by default.

Migration Topics

Each of the following issues will require some level of consideration by InterBase users that are migrating from earlier versions. The relative importance of these issues depends on the types of application programs that the customer has written. For example, if an application did not make use of the journaling function in Version 3.3, then journaling will not be a migration concern.

System Configuration Changes

Unix System Settings

The allowed numbers of Unix system lock headers and semaphores have increased. Semaphores have increased from 32 in Version 3.2 to 128 in Version 3.3, to 64K in Version 4.0. Because the numbers have increased, there will not be an adverse impact on application programs. A given system can run more processes connecting to the database(s) on the server. For complete information on Unix system settings, see the Borland World Wide Web page.

Linking

Full backend libraries (**-lgds_b**) are no longer supported. All programs will need to be run through **gpre** and relinked. That linking process will be different from Version 3.*x*. Compiling and linking are system-specific tasks and complete information is provided as part of your operating system's documentation. For issues specific to InterBase, see the *Programmer's Guide*.

Library Calls

For applications to be portable across platforms, all the **gds_$** calls need to be replaced with **isc_** calls. This requires changes in how some arguments are passed to the InterBase functions. A list of the possible valid **gds_$** calls can be provided, specifically the DSQL calls documented for the Windows Client SDK. All applications written to use the InterBase V3.3 API should be rewritten to use the **isc_** calls. For complete information on the **isc_** calls, see the *API Guide*.

InterBase System Changes

Keywords

Keywords, also called reserved words, are words that are used as part of the InterBase SQL language. Keywords are reserved by SQL and cannot be used in any way by application programs.

InterBase 4.0 includes more reserved words than earlier versions. It is possible that an application program used with Version 3.*x* contains words that have now become keywords. All occurrences of keywords must be found and changed in order for the application program to work. A program is provided that generates an ISQL script that would locate all the occurrences of reserved words used in application programs.

For a complete list of the reserved words, see the *Language Reference*.

Error Codes

The InterBase Error Codes have been made more specific. Earlier versions of InterBase used one numerical value to reflect several different errors. Version 4.0 uses a unique numerical code to identify each of these previously grouped errors. For example, SQLCODE -804 (wrong number of arguments) is still the same. But, previously, multiple rows in a singleton select would also return -804. In Version 4.0 it will return -811.

Applications that depend on specific error code values (for example, detecting and branching on error code values) may need a change to take the new values into account. For complete lists of the InterBase and SQLCODE error codes for Version 4.0, see the *Language Reference*.

gpre

The Embedded SQL preprocessor (**gpre**) for Version 4.0 supports the use of GDML code, though no additional enhancements were made for Version 4.0. It is recommended that any GDML statements be changed to SQL.

New Features for Version 4.0

SQL Descriptors

The XSQLDA is a host-language data structure that DSQL uses to transport data to or from a database when processing an SQL statement string. All DSQL applications must declare one or more extended SQL descriptor areas (XSQLDAs). The XSQLDA structure definition can be found in the *ibase.h* header file in the InterBase *include* directory.

Use of the Extended SQL descriptor areas (XSQLDAs) is the default in Version 4.0. C++ will not be able to compile existing programs that use the SQLDA structures. Those programs will have to be rewritten to call XSQLDA structures.

gbak Switches

The meaning of the **-u** switch to **gbak restore** has changed. In Version 3.*x*, the **-u** switch meant (**u**)**se_all_space**. In Version 4.0, the **-u** switch means (**u**)**sername** and is used with either backup or restore to force InterBase to check that the user has sufficient privileges to perform the operation.

gbak restore now allows the use of multiple-character switches. Automatic **cron** jobs running **gbak restore** must be changed to use the multiple-character switch.

gbak -y has changed. When specifying that the **gbak** status messages be sent to a file, that file must not already exist. If you specify a file that already exists you will get an error message and the **gbak** will not complete.

gbak, gfix, and gsec Switches

gbak, **gfix**, and **gsec** now employ several multiple-character switches. Previously, all the switches for **gbak**, **gfix**, and **gsec** could be stated as a single character (for example, **gbak -k -n -o**). In Version 4.0 some switches require more than one character (for example, **gbak -c -pa**). Application programs written for Version 3.*x* that include **gbak**, **gfix**, and **gsec** should be checked and possibly rewritten to make sure that they work correctly with multiple-character switches.

Shadowing

The operation of shadows in InterBase Version 4.0 on Unix is similar to the operation in Version 3.3. There have been several bug fixes that will improve the operation of shadows.

Version 3.*x* GDML Features Unsupported in Version 4.0

GDML Features

The triggers, COMPUTED BY fields, views, and validation check constraints were implemented in GDML in Version 3.*x*. Applications that use triggers, views, and validation should be rewritten for any Version 4.0 applications.

SQL Select Statement

Version 3.*x* used GDML Record Selection Expressions (RSEs) which are equivalent to the SELECT statement in SQL. It is a quick way to access a record that has already been selected. Applications that use GDML RSEs must be rewritten to use SELECT.

Access Privileges

In Version 3.*x*, triggers use the RDB$USER field from the RDB$USER_PRIVILEGES system table to stamp a record with the user name of the user changing the record, or to check to see if this user is allowed to perform **insert, modify**, or **erase** operations on a specific table. These security functions in Version 4.0 are performed using the SQL GRANT statement. All the RDB$USER stamps must be recast as GRANT statements. For complete information on the GRANT statement, see the *Programmer's Guide.*

Components Not Supported in Version 4.0

qli

The Query Language Interpreter (**qli**) is not supported in InterBase V4.0. Applications that use **qli** should be rewritten in a 3GL language (C, C++, Pascal, etc.) or find another 4GL front-end. All **qli** procedures or scripts will have to be rewritten as Interactive, Dynamic, or embedded SQL programs.

pyxis and fred

The **pyxis** and **fred** libraries are supplied with Version 4.0 but are no longer supported. Programs that make calls to these libraries should be re-written, preprocessed and linked.

Journaling

The Version 3.*x* journaling function is no longer supported.

User Response

The development of the list of migration issues is an ongoing process. We encourage feedback. If any application programmer encounters an issue that affects migration, we would appreciate being informed.

CHAPTER 4

Using InterBase on Unix

This chapter provides special notes on using InterBase on Unix.

InterBase for Unix enables you to issue interactive SQL (**isql**) commands from the Unix command line. For information on using command-line **isql**, see Chapter 5: "Using Command-line isql."

Database Administration

You can perform database administration (DBA) for InterBase using the command-line DBA utilities on the server or using the Server Manager on the remote client. For information on using the command-line DBA utilities, see Chapter 6: "Using the Command-line DBA Utilities." For information on using the Server Manager, see the *Windows Client User's Guide*.

Working With Databases

When working in networked client/server environments, keeping the server's and client's versions of InterBase synchronized is crucial. Remember that, even though you are working on the client, the server's version of the software is the one that is running. Make sure you check both versions.

InterBase does not place any restrictions on the names of files beyond those placed by Unix itself. Because InterBase will often be used in environments which include DOS machines, whenever possible use names that meet the DOS file name restrictions: an eight letter name with a maximum three letter extension.

Forced and Buffered Writes

By default, InterBase performs *buffered writes* (also referred to as asynchronous writes). Unlike a *forced write*, when InterBase performs buffered writes, it does not physically write data to disk until a pre-specified event occurs. That event can be when a certain amount of information has been collected for a write, an associated event has occurred, or a certain time interval has elapsed.

If forced writes are not enabled, then even though InterBase performs an internal write, the data may not be physically written to disk, because the operating system buffers 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 will slow performance. In particular, operations which involve data modification will be slower.

You can also enable and disable forced writes with the **gfix** command-line DBA utility. For more information on **gfix**, see Chapter 6: “Using the Command-line DBA Utilities.”

Shutdown and Logoff

If the Unix server is shut down or if a user logs off while there is an InterBase application running, data can be lost. If you shut down or log off during database activity, then the active database can get orphan pages.

You can mend orphan pages with the **gfix** command-line DBA utility. For more-information on **gfix**, see Chapter 6: “Using the Command-line DBA Utilities.”

Building Applications

You can create Unix applications for InterBase using the **gpre** precompiler to create C programs from programs containing embedded SQL commands. You can also build applications using the InterBase API.

For general information on creating embedded applications, see the *Programmer’s Guide*. For information on creating applications with the InterBase API, see the *API Guide*.

Using the gpre Precompiler with C++

The precompiler, **gpre**, will append the file name extension, *.cxx*, when it creates C++ output files. For example, if an embedded C++ source file is called:

```
foo.exx
```

then **gpre** creates an output file called:

```
foo.exx.cxx
```

Creating User-defined Functions and BLOB Filters

User-defined functions (UDFs) and BLOB filters are Unix shared libraries. These libraries must always reside on the server.

There are two ways you can run such functions:

- Locally, when you are building and running an application on the server.
- Remotely, when you are building and running an application on an NT client that will use InterBase on a remote server.

To run a function locally, you must state a fully-qualified path in the declaration to the database.

CHAPTER 5

Using Command-line isql

This chapter describes the command-line **isql** utility (interactive SQL) available with Unix servers.

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 chapter 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 Appendix B: “isql Command Reference.”

Invoking isql

You can use **isql**:

- Interactively to process SQL statements, by entering statements at the **isql** prompt.
- Non-interactively to process SQL statements in a file.

To start the **isql** utility, type the following at a Unix 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** will start interactively or non-interactively, depending on the options. For example, reading an input file or writing to an output file are non-interactive tasks, so the **-input** or **-output** options do not start an interactive session. Additional non-interactive options include **-a**, **-database**, **-extract**, and **-x**, which are used when extracting DDL statements.

When using **isql** interactively, the following prompt will appear:

```
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 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.

Table 5-1: **isql** Command-line Options

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 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.
-e[**cho**]	Displays (echoes) each statement before executing it.
-ex[**tract**]	Same as **-x**.
-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.
-n[**oauto**]	Turns off automatic commitment of DDL statements. By default, DDL statements are committed automatically in a separate transaction.
-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.

Table 5-1: **isql** Command-line Options (Continued)

Option	Description
-p[**assword**] *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.
-pag[**elength**] *n*	Prints column headers every *n* lines instead of the default 20.
-t[**erminator**] *x*	Change 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**.

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, *hera*, accepts the specified user and password combination:

```
isql -user sales -password mycode hera:/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 Unix. Use the **connect** command with the full path of the database as the argument. For example:

  ```
  connect /usr/interbase/examples/v4/employee.gdb;
  ```

- A remote database on an NT, Unix, or NetWare 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. On a Unix platform, precede each directory or file name with a slash (/). On an NT platform, precede each directory or file name with a slash (/) or a backslash (\).

  To connect to a database on a Unix platform named *node1*:

  ```
  connect node1:/usr/interbase/examples/v4/employee.gdb;
  ```

  To connect to a database on an NT platform named *node2*:

  ```
  connect node2:/users/interbase/examples/v4/employee.gdb;
  ```

Note Be careful not to confuse node names and shared disks, because both are specified with a colon separator. If you specify a single letter that maps to a disk drive, then it will be assumed to be a drive, not a node name.

Transaction Behavior in isql

When you start **isql**, InterBase begins a transaction. That transaction will be 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 will automatically start a new transaction, or you can start a transaction with the SET TRANSACTION statement.

Note **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 **isql -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.

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:

- Generators.
- UDF code and BLOB filters, because they are 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.

Order of Extraction

The **isql -extract** option extracts all SQL-based metadata in the following order:

Table 5-2: Order of Extraction

Metadata	Comments
database	Extracts database with default character set and **page_size**.
domains	
tables	
BLOB data types and known subtypes	
NULL and default values	
PRIMARY KEY constraints	
CHECK constraints	
FOREIGN KEY constraints	Must be added after tables by ALTER TABLE to avoid tables referenced before being created.
indexes	Only for tables extracted, except triggers from referential or unique constraints.
views WITH CHECK OPTION	
stored procedures	In the extracted DDL, stored procedures are created with no body and then ALTER PROCEDURE adds the text of the procedure body.
triggers	Does not extract triggers from CHECK constraints.
GRANTs	From RDB$USER_PRIVILEGES table.

For example, the following statement extracts the system catalogs from the database *employee.gdb* to a file called *employeeout.gdb*:

```
isql -extract -output employeeout employee.gdb;
```

The resulting output script is created with **-commit**s 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. For a detailed description of each command, see Appendix B: "isql Command Reference."

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*. For the full syntax of the SHOW commands, see Appendix B: "isql Command Reference."

SHOW commands operate on a separate transaction from user statements. They run as READ COMMITTED background statements and acknowledge all metadata changes immediately.

The following table lists **isql** SHOW commands:

Table 5-3: **isql** SHOW Commands

Command	Description
SHOW CHECK	Shows all CHECK constraints for the named object.
SHOW DATABASE	Shows the name and files of the current database, and the relevant statistics for the database.
SHOW DOMAINS	Lists all domains defined in the database or displays information about a named domain.
SHOW EXCEPTIONS	Lists all exceptions defined in the database or displays information about a named exception.
SHOW FILTERS	Shows all filters defined in the database or displays information about a named filter.
SHOW FUNCTIONS	Lists all UDFs defined in the database or displays information about a named UDF.
SHOW GENERATORS	Lists all generators declared to the database or displays information about a named generator.
SHOW GRANT	Shows all permissions for the named object.
SHOW INDEX	Lists all indexes in the tables in the database.
SHOW PROCEDURES	Lists all procedures defined in the database or displays the text and parameters of a specified procedure.
SHOW SYSTEM	Shows a list of system tables (the metadata).
SHOW TABLES	Lists the columns and types, constraints, and triggers defined for a given table. Includes the view definition for views.
SHOW TRIGGERS	Lists all triggers defined in the database or displays information for the named trigger, including trigger text.
SHOW VERSION	Displays the database and access method version.
SHOW VIEWS	Lists all views defined in the database or displays information about a specified view.

SET Commands

SET commands enable you to view and change the **isql** environment. The following table lists the SET commands:

Table 5-4: **isql** SET Commands

Command	Description
SET	Lists which SET options are on.
SET AUTODDL	Toggles the commit feature for DDL statements.
SET BLOBDISPLAY *n*	Turns on the display of BLOB type *n*. *n* is required for the BLOB types to be displayed.
SET COUNT	Toggles the count of selected rows on or off.
SET ECHO	Toggles the display of each command on or off.
SET LIST *string*	Displays columns vertically or horizontally.
SET NAMES	Specifies the active character set.
SET PLAN	Determines whether or not to display the optimizer's query plan.
SET STATS	Toggles the display of the performance statistics on or off.
SET TERM *string*	Allows you to change to an alternate terminator character(s).
SET TIME	Toggles display of time in DATE values.

By default all settings are initially OFF except AUTODDL, and the terminator is a semicolon (;). Each time you start an **isql** session, it begins with the default settings.

Tip If you have a group of settings you commonly use, you can keep them in a file to be input at the beginning of the **isql** session.

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 following table lists all other **isql** commands:

Table 5-5: **isql** Miscellaneous Commands

Command	Description
BLOBDUMP	Places the contents of a BLOB column into a file.
/* Comment */	Used to comment **isql** script files.
EDIT	Calls the system editor. Allows you to edit a file without exiting **isql**.

Table 5-5: **isql** Miscellaneous Commands (Continued)

Command	**Description**
EXIT	Commits the current transaction, closes the database, and ends the **isql** session.
HELP	Displays a brief help message on **isql** commands.
INPUT	Reads and executes commands from the named file without prompting the user. The INPUT command can be nested.
OUTPUT	Directs output to the named file or back to the standard output.
QUIT	Issues the QUIT command to exit **isql**. All uncommitted work is rolled back.
SHELL	Enables you to enter Unix operating-system commands.

BLOBDUMP

BLOBDUMP places the contents of a BLOB column in a named file for reading or editing. The syntax is:

```
BLOBDUMP blob_id filename
```

The BLOB ID is always displayed in query output as the column value. Contents of BLOB columns are displayed at the end of all columns if SET BLOBDISPLAY is turned on.

All BLOB IDs are two hexadecimal numbers separated by a colon (:). The first number is the table ID of the table containing the BLOB, and the second number identifies the BLOB.

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.

Comments

isql script files can be commented exactly like C programs:

```
/* comment */
```

A comment can be of any length: less than one line or many lines, as long as it is preceded by "/*" and followed by "*/".

EDIT

EDIT can be used to edit and execute commands in an SQL file or commands in the current **isql** session. Make sure to set up a default Unix editor for your system before issuing this command.

After exiting the editor, **isql** automatically executes the commands in the edited buffer.

The EDIT command does not work in input files.

SHELL

The SHELL command provides temporary access to Unix operating system commands. Use SHELL to execute an operating-system command without ending the current **isql** session.

SHELL with no argument temporarily interrupts the **isql** session and returns you to the Unix prompt. Type EXIT to return to **isql**. Follow SHELL with a Unix command to execute the Unix command immediately. The **isql** prompt will return.

SHELL does not commit transactions.

INPUT

INPUT reads the file given as its argument and executes SQL statements in the file. In this way, INPUT enables execution of commands without prompting. The file must contain legal **isql** statements. It is often advantageous to write DDL statements in a file using a text editor, to make the authoring of the metadata objects easier. You can then use INPUT to read the file into **isql** and define the objects.

The EDIT command does not work in input files.

INPUT is particularly useful when working with stored procedures and triggers. It allows you to use a text editor to edit the file containing CREATE PROCEDURE or TRIGGER statements, and then use INPUT to read the file into **isql**, where you can test the procedure or trigger.

Tip If you are working with a database object that is updated frequently, put DROP *objectname* as the first line of code to avoid error messages because duplicate names are not allowed in a database.

Exiting isql

To exit the **isql** utility and roll back all uncommitted work, enter:

```
QUIT;
```

To exit the **isql** utility and commit all work, enter:

```
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:

Table 5-6: SQLCODE and Message Summary

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.

For a detailed discussion of error handling, see the *Programmer's Guide*. For a complete listing of SQLCODE and InterBase status array codes, see the *Language Reference*.

CHAPTER 6

Using the Command-line DBA Utilities

This chapter describes the command-line database administration utilities available with InterBase. These utilities provide a Unix command-line interface for performing database administration tasks.

The command-line DBA utilities are:

- **gsec**: For administering security for InterBase servers.
- **gfix**: For maintaining databases, including sweeping, repairing, shutting down, and restarting databases, and recovering “limbo” transactions.
- **gbak**: For backing up and restoring databases.

Using the Security Utility

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.

The security database, *isc4.gdb*, is installed in the InterBase directory (by default, */usr/interbase*). To attach to a database on the server, users must specify a user name and password. The user name and password are verified against information stored in *isc4.gdb*. If a matching row is found, the attachment succeeds.

Important You must log in to Unix as root to use **gsec**.

The Security Database

Every user of an InterBase server requires an entry in the *isc4.gdb* security database. The security utility, **gsec**, lets you display, add, modify, or delete information in *isc4.gdb*.

The following table describes the contents of *isc4.gdb*:

Table 6-1: Format of the *isc4.gdb* Security Database

Column	Required?	Description
User name	Yes	Name the user supplies when logging in
Password	No (but highly recommended)	User's password
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)

To use **gsec** interactively, type "gsec" at the Unix prompt. The prompt will then change to GSEC>, indicating that you are in interactive mode. To quit an interactive session, type "quit."

Security Utility Commands

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

Table 6-2: Summary of **gsec** Commands

Command	Description
di[**splay**]	Displays all rows of *isc4.gdb*.
di[**splay**] *name*	Displays information only for user *name.*
a[**dd**] *name <data>*	Adds user *name* to *isc4.gdb*. *<data>* specifies the columns and their associated values. Supply *<data>* by specifying various options. For information, see "Adding Entries to the Security Database," in this chapter.
mo[**dify**] *name <data>*	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.

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 name    uid   gid
```

```
------------ ----- -----
FRED         123
BARNEY       123
BETTY        123
```

Note that passwords are never displayed.

Adding Entries to the Security Database

To add users to the security database, use the **add** command and supply at least a user name and password.

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
------------ ----- -----
       JONES   0     0
```

In the previous example, the **-pw** option indicates that the next argument is the password. You can supply additional options, with corresponding arguments for other information in the security database. The following table summarizes the **gsec** options. For each option, the initial letter or letters are required and optional parts are enclosed in brackets.

Table 6-3: **gsec** Options

gsec Option	Description
-pw	User's password (string)
-u[**id**]	User ID (integer)
-g[**id**]	Group ID (integer)
-f[**name**]	User's first name (string)
-mn[**ame**]	User's middle name (string)
-l[**name**]	User's last name (string)

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
```

The user name entry in *isc4.gdb* is case insensitive, but **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 the deletion with the **display** command.

Using the Security Utility from the Unix Command Line

To use **gsec** from the Unix command line, precede each command with **gsec** and prefix each **gsec** command with a hyphen (-). For example, to assign password, "sesame" to user, ALADDIN, enter the following at the command line:

```
> gsec -add ALADDIN -pw sesame
```

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

```
> gsec -display
```

Using the Database Maintenance Utility

The database maintenance utility is **gfix**. You can use **gfix** to perform a variety of database maintenance tasks, including sweeping, repairing, shutting down, and restarting databases, and recovering "limbo" transactions.

The general syntax for **gfix** is:

```
gfix [options] db_name
```

Each database maintenance task has associated options, listed in the corresponding sections that follow.

Sweeping a Database

Sweeping a database systematically removes outdated records from the database. Periodic sweeping prevents a database from growing too large. Sweeping a database too frequently can also slow system performance.

The **gfix** utility provides control over database sweeping. You can:

- Change the automatic sweep interval.
- Disable automatic sweeping.
- Sweep a database at specified times.

Note Sweeping a database does not require you to shut it down. This allows you to schedule sweeping when it will least affect users (for example, by running it at a low priority or at off-peak hours).

Overview of Sweeping

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

To limit the growth of the database, InterBase performs *garbage collection*, which frees up space allocated to outdated versions of a record. Whenever a transaction accesses a record, the server eliminates outdated versions ("garbage"). Records that were rolled back are ignored by typical transactions and will not be collected. To guarantee that all records are collected, including those that were rolled back, InterBase periodically "sweeps" the database.

When sweeping a database, InterBase reads every record in the database. This forces garbage collection of outdated record versions as well as rolled back records. InterBase automatically sweeps a database every 20,000 transactions by default.

Because automatic sweeping is tied to a transaction, sweeping a database can affect application performance. Because InterBase is a multi-threaded server, the sweep can spawn a separate process so that it does not block other processes.

Note Sweeping a database is not the only way to perform systematic garbage collection. Backing up a database achieves the same result because the backup utility, **gbak**, must read every record, forcing garbage collection throughout the database. As a result, regularly backing up and restoring a database can reduce the need to sweep. You can maintain better application performance this way.

Options for Database Sweeping

Use the following options to perform sweeping with **gfix**.

Table 6-4: Sweeping Options

Option	Description
-s[**weep**]	Force an immediate sweep of the database. Useful if automatic sweeping is disabled. Exclusive access is not necessary.
-h[**ousekeeping**] *n*	Change automatic sweep interval to *n* transactions, or disable sweeping by setting *n* to 0. Default interval is 20,000 transactions. Exclusive access is not needed.

Changing the Sweep Interval

To change the automatic sweep interval, use the **-housekeeping** (**-h**) option. For example, you can set the sweep interval to 10,000 transactions as follows:

```
gfix -h 10000 employee.gdb
```

As the DBA, you should determine the sweep interval that provides the best database performance.

On one hand, sweeping more often can reduce the time for transaction startup. Sweeping a database can affect transaction startup if rolled back transactions exist in the database. As the time increases since the last sweep occurred, the time for transaction startup may also increase.

On the other hand, frequent database sweeps may reduce application performance. This might occasionally create a delay in transaction startup.

Note 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 DBA to fine-tune the database by disabling sweeping and scheduling it for specific times.

Disabling Automatic Sweeping

To disable automatic sweeping, use the **-housekeeping** option to set the sweep interval to 0. For example,

```
gfix -h 0 employee.gdb
```

Disabling automatic sweeping is useful if:

- Maximum throughput is important. Transactions will never be delayed by sweeping.
- You want to schedule sweeping at specific times. After automatic sweeping is disabled, you can manually sweep the database at a specified time.

Sweeping a Database at a Specified Time

To sweep a database immediately, use the **-sweep** (**-s**) option. For example,

```
gfix -s employee.gdb
```

Because sweeping a database does not require shutting it down, you can schedule sweeping when it will least affect users by running the process at a low priority or during off-peak hours. Use the **cron** command to schedule a sweep at a specified time.

For example, the following command will run a sweep on *myserver* at 3:00 a.m.:

```
cron //myserver 03:00 "gfix -s employee.gdb"
```

Note You must log in as root to use the **cron** command.

Repairing a Database

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 (killed), committed data is preserved, and uncommitted changes are rolled back. If InterBase has already assigned a database 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 result in "broken" or "lost" data structures, such as a database page or index. These corrupt data structures can prevent recovery of committed data.

The **gfix** database maintenance utility provides several options for minor repair of data structures, including:

- Validating a database.
- Mending the database if corruption is reported.

Options for Database Repair

Use the following options when performing database repair with **gfix**:

Table 6-5: Database Repair Options

Option	Description
-v[**alidate**]	Locate and release pages that are allocated but unassigned to any data structures. Also reports corrupt structures.
-f[**ull**]	Used with **-validate** to check record and page structures, releasing unassigned record fragments.
-i[**gnore**]	Ignore checksum errors when validating or sweeping.
-m[**end**]	Mark corrupt records as unavailable, so they are skipped (for example, during a subsequent back up).
-n[**o_update**]	Used with **-validate** to report corrupt or mis-allocated structures. Structures are reported but not fixed.

Validating a Database

Validating a database means verifying the integrity of data structures. Validate a database:

- Whenever a database backup is unsuccessful.
- Whenever an application receives a "corrupt database" error.
- To periodically monitor for corrupt data structures or mis-allocated space.

- When data corruption is suspected.

Validation is performed with the **-validate** (**-v**) **gfix** option. This will report corrupt data structures and mis-allocated database pages, and return orphan pages to free space.

In the following example, the first **gfix** command reports orphan pages. The second command confirms that the orphan pages were freed, indicating that the database structures were not "broken," merely unallocated:

```
> gfix -validate -full emp.gdb
  Page 142 is an orphan
  Page 146 is an orphan
  Page 150 is an orphan
  Page 152 is an orphan
> gfix -validate -full emp.gdb
```

In this example, the **-full** option modifies **-validate**. By itself, **-validate** reports and releases only page structures. When combined with **-full**, the **-validate** option reports and releases record structures as well as page structures. Use **-full** when you want to verify all structures.

By default, validating a database updates it, if necessary. To prevent updating, use the **-no_update** option after **-validate**. The following example reports structures but does not fix them:

```
gfix -validate -no_update emp.gdb
```

Mending a Corrupt Database

In addition to reporting mis-allocated structures, **-validate** reports broken structures caused by write errors in the operating system or hardware. The **-validate** option does not fix write errors; it only reports them.

To fix write errors, use the **-mend** (**-m**) option. **-mend** fixes problems that cause records to be corrupt, reports errors, and marks corrupt structures. In subsequent operations (such as backing up), InterBase ignores the marked records.

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 InterBase backup utility (**gbak**), because it cannot back up a database containing corrupt data. For example, enter:

   ```
   cp my.gdb my_copy.gdb
   ```

2. Run **gfix -mend** to mark corrupt structures in the database copy. If **-full** is also specified, **gfix** marks both record and page structures (for exam-

ple, damaged BLOB data is marked); if **-full** is not specified, **gfix** marks only page structures. **gfix** reports the errors it finds:

```
gfix -mend -full my_copy.gdb
   Chain for record 9 is broken in table JOB (16)
   Page 32 is an orphan
   Page 196 is used but marked free
   Page 197 is used but marked free
```

If **gfix -mend** reports any checksum errors, reissue the command using the **-ignore** option in addition to **-mend**:

```
gfix -mend -full -ignore my_copy.gdb
```

3. Run **gfix -validate -full** to see if the errors reported by **gfix -mend** are actually fixed:

   ```
   gfix -validate -full my_copy.gdb
      Record 9 is marked as damaged in table JOB (16)
      Page 32 is an orphan
      Page 33 is an orphan
   ```

   Note that the free pages are no longer reported, and the broken record is marked as damaged. Any records marked by **-mend** are ignored when the database is backed up.

4. Back up the mended database with **gbak**:

   ```
   gbak -b my_copy.gdb my_copy.gbk
   ```

   At this point, any damaged records are lost, because they were not included during the backup.

5. Restore the database to rebuild indexes and other database structures:

   ```
   gbak -c my_copy.gbk new_copy.gdb
   ```

   The restored database should now be free of corruption.

6. Verify that restoring the database fixed the problem:

   ```
   gfix -validate -full new_copy.gdb
      <No messages reported>
   ```

In the previous example, **gfix** was able to mend the database. You can therefore delete the original corrupt database, *my.gdb*.

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

Handling Checksum Errors

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, follow the procedure in the previous section but use the **-ignore** option in addition to **-mend**. For example:

```
gfix -mend -full -ignore my_copy.gdb
```

The **-ignore** option enables the InterBase backup utility (**gbak**) to ignore checksums when backing up a database. Ignoring checksums allows successful backup of a corrupt database, but the affected data may be lost.

Caution 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.

Recovering Limbo Transactions

The database maintenance utility enables you recover limbo transactions. A limbo transaction can be caused by hardware failure, and recovering it means the transaction is committed or rolled back. The DBA can decide whether to commit or roll back a specific transaction, or you can let **gfix** decide. This process of resolving limbo transactions to make them commit or roll back is called *automated transaction recovery.*

The Two-phase Commit Process

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.

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 phase two, some subtransactions will be committed and others will not be. 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 do not know whether to commit their changes or roll them back. You can resolve, or recover, these limbo transactions using the database maintenance utility.

Automated Transaction Recovery

The database maintenance utility analyzes the state of subtransactions by determining when the two-phase commit failed. If the first transactions are in limbo but later transactions are not, the maintenance utility assumes that the prepare phase did not complete. In this case, **gfix** prompts you to roll back.

If the first transactions are missing and later transactions are in limbo, **gfix** assumes that the prepare phase completed but the commit phase did not. In this case, **gfix** prompts you to commit the transaction.

If all transactions are prepared, **gfix** assumes that the failure occurred between phase one and phase two of the two-phase commit. In this case, you must decide whether to commit or roll back the transaction.

Options for Transaction Recovery

gfix provides several options to use in automated transaction recovery:

Table 6-6: Transaction Recovery Options

Option	Description
-list	Lists the ID number of each transaction in limbo, including the partner transaction if it is a multi-database transaction; also lists the current state and the action that automated recovery would take for a multi-database transaction.
-prompt	Lists limbo transactions and prompts you for action. Must be used with **-list**.
-commit *id* | **all**	Commits a single transaction specified by *id* or commits all limbo transactions. **gfix** advises against a commit if some of the transactions were not prepared.
-rollback *id* | **all**	Rolls back a single transaction specified by *id* or rolls back all limbo transactions.
-two_phase *id* | **all**	Performs automated transaction recovery on a single transaction *id* or on all limbo transactions. Commits or rolls back the transaction, depending on its state. If all transactions are prepared but you did not specify a commit or rollback, **gfix** prompts you for action.

To recover limbo transactions, use the following procedure:

1. List the transactions in limbo:

   ```
   gfix -list database
   ```

2. Perform one of the following four actions:

   - Commit a specified limbo transaction or all limbo transactions. For example,

     ```
     gfix -commit all database
     ```

   - Roll back a specified limbo transaction or all limbo transactions. For example, to roll back transaction 766 (as determined by **-list**), enter the following:

     ```
     gfix -rollback 766 database
     ```

   - Perform automated two-phase recovery. Limbo transactions will be committed or rolled back, depending on the state of the transaction. If **gfix** cannot determine the state of the transaction (for example, because failure occurred between phase 1 and phase 2), **gfix** prompts you to commit, roll back, or ignore the transaction. For example,

     ```
     gfix -two_phase all database
     ```

   - Use **-prompt** so that **gfix** prompts you for action. This is useful when you want to decide what to do, one transaction at a time. For example,

     ```
     gfix -list -prompt database
     ```

Shutting Down a Database

Maintaining a database often involves shutting it down. *Shutting down* a database means that no process can attach to it except the SYSDBA user. When this occurs, SYSDBA has *exclusive access* to the database. Only SYSDBA or the owner of a database can shut it down.

When a database is shut down, the user who shut it down has *exclusive access* to the database. Exclusive access to a database is required to:

- Validate the database
- Add a foreign key to a table in the database or drop a foreign key from a table in the database.
- Add a secondary database file.

Shutdown Options

To shut down a database, use the **gfix -shut** command as follows:

```
    gfix -shut method timeout database
```

Table 6-7: Shutdown Options

Option	Description
method	One of three additional **gfix** options: **-attach** prevents any new attachments to the database. All existing database attachments can complete their operations unaffected. The database is shut down after all current attachments detach from the database. **-tran** prevents any new transactions from starting but lets existing transactions finish. After transaction processing is disabled, the database is shut down. **-force** shuts down the database when: • There are no connections to the database or • At the end of the timeout period If the timeout period expires and there are still processes connected to the database, **gfix** will roll back all current transactions and shut down the database. This option forces shutdown of the database and should be used with caution.
timeout	Timeout period in seconds. With **-attach**, the database will be shut down when there are no active connections during the timeout period. If there are still processes connected at the end of the timeout period, the shutdown is canceled. With **-tran**, the database will be shut down when there are no active transactions during the timeout period. If there are still active transactions at the end of the timeout period, the shutdown is canceled. With **-force**, the database will be shut down as soon as there are no processes connected to the database or at the end of the timeout period. Minimum timeout is 0 (immediate shutdown); maximum is 32,767 seconds (about 9 hours).
database	Name (and full directory path) of the database to shut down.

Important **gfix** does not notify users of an impending database shutdown. The DBA is responsible for notifying users in an appropriate way. Send mail or broadcast a message to all users of an impending shutdown.

Preventing Further Attachments

Use the **-attach** option to prevent any attachments to the database until it is shut down.

For example, suppose you need to shut down the database, *sales.gdb*, at the end of the day (five hours from now), but the marketing staff are using the database to generate important sales reports. In this case, shut down *sales.gdb* by entering the following command:

```
gfix -shut -attach 18000 sales.gdb
```

This command indicates a maximum five-hour wait (18,000 seconds) before shutting down *sales.gdb.* Any users who are already attached to the database will be able to finish processing their sales reports, but no further attachments will be allowed. If all attachments are closed any time during the five hours, then the database will be shut down immediately.

It would be inappropriate to use **-tran** as the method of shutdown in this case. Generating a report could require several transactions, and if you use **-tran**, a user might be disconnected from the database before completing all transactions necessary to generate the report.

Preventing Further Transactions

Use the **-tran** option to prevent new transactions during the timeout period. If any transactions are still active at the end of the timeout period, then the shutdown is canceled.

For example, suppose you need to shut down the database, *orders.gdb*. This database is used continuously by dozens of customer service representatives to enter new orders and query existing orders.

To shut down *orders.gdb* in this case, using **-attach** would have no effect. The **-attach** option prevents only new attachments; it does not affect current attachments. If users plan to remain attached to a database for a long time, use the **-tran** option to shut down the database. For example,

```
gfix -shut -tran 3600 orders.gdb
```

This command specifies to wait up to an hour wait (3,600 seconds) before shutdown. During that time, users cannot start any new transactions. By using **-tran**, you avoid waiting for users to detach. Instead, current users complete a logical unit of work before losing database access.

Note The **-tran** option implies the use of **-attach** because attachments, like other database activities, are started by a transaction. So when **-tran** prevents the start of new transactions, **-tran** also prevents new attachments. As soon as all current transactions have finished, the database is shut down.

Forcing Shutdown

With the **-tran** or **-attach** options, the database will not be shut down if there are active transactions or connections, respectively, at the end of the timeout period. In some cases, it will be necessary to shut down the database anyway.

Use the **-force** option to force a shutdown of a database. Provide the timeout period, in seconds, as the argument to the option. The **-force** option will shut down the database:

- As soon as there are no active transactions or connections to the database, or
- At the end of the timeout period, whichever comes first.

When one of these two condition occurs, all transactions will be rolled back and all processes will be disconnected from the database.

Note As long as there are processes connected to the database, **gfix** will not shut down the database until the timeout period expires. In this case, other processes can initiate new transactions and connections to the database during the timeout period.

For example, suppose an emergency requires the immediate shut down of *orders.gdb*. To do this, use the following command:

```
gfix -shut -force 600 orders.gdb
```

This command indicates a ten-minute wait (600 seconds) before detaching all processes from the database. Any transactions that are in progress at this time will be rolled back. The **-force** option does not prevent new attachments or new transactions.

Important Because the **-force** option will interfere with normal database operation, use it only in emergencies, and be sure to provide appropriate notification to users.

Restarting a Database

After a database is shut down, it must be restarted (brought back online) before users can access it. If you shut down a database, use the following syntax to restart it:

```
gfix -online database
```

where *database* is the name of the database that was previously shut down.

Note The **-online** option can also be used to cancel a shutdown command that was issued but has yet to take effect.

Controlling Performance of Forced Writes

By default, the InterBase Workgroup Server for Unix performs *buffered writes* (also referred to as asynchronous writes). Unlike a *forced write*, when InterBase performs buffered writes, it does not physically write data to disk until a pre-specified event occurs. That event can be when a certain amount of information has been collected for a write, an associated event has occurred, or a certain time interval has elapsed.

If forced writes are not enabled, then even though InterBase performs an internal write, the data may not be physically written to disk, because the operating system buffers 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 will slow performance. In particular, operations which involve data modification will be slower.

You can also enable and disable forced writes with the **gfix** command-line DBA utility. For more information on **gfix**, see Appendix C: "Command-line DBA Utilities Reference."

To enable or disable forced writes, use the following command:

```
gfix -w database
```

This command toggles the performance of forced writes. If forced writes are on, it disables them. If forced writes are off, it enables them.

Using the Backup and Restore Utilities

To guard against disk crashes, power failure, or other potential data loss, databases should be backed up regularly. When a database is restored from a backup file, it will be re-created in its condition at the time of the backup. The command-line utility for performing backup and restoration is **gbak**.

Using **gbak** provides these advantages:

- Database performance can be improved.

  Backing up and restoring a database garbage-collects outdated records and balances indexes. The process also frees disk space occupied by deleted records and packs the remaining data, reducing database size.

When you restore, you can change the database page size or distribute the database among multiple files or disks.

- Backups can run concurrently with other users.

  You need not shut down the database to run a backup. Any data changes that occur after the backup begins are not recorded in the backup file. After you create a database backup, you can include it as part of a regular system backup.

- Multi-file databases are never partially backed up.

  If a database spans multiple files, **gbak** backs up either all files or none.

- Data can be transferred to another operating system.

  You can back up a remote database across a network. If desired, you can also make a backup in a generic format. This is called a *transportable backup* and allows restoration on a non-networked machine. Making transportable backups is highly recommended in heterogeneous environments, whether networked or not.

General Syntax

To run **gbak**, specify a source, a target, and any options desired. The syntax is:

```
gbak [options] source target
```

When backing up a database, *source* names an existing database file, and *target* names a destination backup file or device. For examples of backing up, see the next section.

When restoring a database, *source* names an existing backup file or device, and *target* names one or more database files to restore from the backup. For examples of restoring, see "Performing a Simple Restoration," in this chapter. For examples of restoring to multiple files, see "Splitting a Database into Several Files," in this chapter.

Some *options* are used only for backing up, some only for restoring, and some for either. Specify the full option name or any shorter form down to the least significant abbreviation. For example, you can type **-create_database**, **-create**, or **-c**.

Performing a Simple Backup

To perform a simple backup, use the **-backup_database** (**-b**) option. For example, the following command creates a backup of *employee.gdb* called *employee.gbk*:

```
gbak -backup employee.gdb employee.gbk
```

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

The **-backup** option is not required; if it is omitted, **gbak** performs a backup by default. For clarity however, examples in this guide use **-backup**.

When creating a backup file, **gbak** always stores the database as one file. For example, you cannot split a large database among multiple backup files. Typical backup files will occupy less space than the database because backup files include only the current version of data and need less overhead for data storage.

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

If a database spans multiple files, specify only the first file (the *primary* file) as the source. **gbak** uses the header page of each file to locate additional files, so the entire database can be backed up based on the primary file.

Backing Up Remote Databases

In a remote backup, specify the name of the machine where the database resides. The target backup file must be on the local machine. For example, if a database resides on a remote machine named *zeus*, back it up as follows:

```
gbak -backup zeus:/usr/db/employee.gdb employee.gbk
```

After creating *employee.gbk* on the local machine, restore it as follows:

```
gbak -create employee.gbk employee.gdb
```

After you move a database from one machine to another, two copies of the database exist. If you intend to maintain only one active copy, delete one.

In the previous example, a colon (:) is used to separate the remote machine name from the remote path name. This indicates that the network connection is via TCP/IP.

Backing Up to a Storage Device

When backing up to a storage device (tape, for example), the tape might become full. If this happens, **gbak** pauses and displays the following prompt:

```
Done with volume # number, "device"
   Press return to reopen that file, or type a new name
   followed by return to open a different file.
Name: default_device
```

The actual volume number and default device name will appear. To continue backing up to the same device, insert a new tape and press **Return**. To continue backing up to another device, insert a tape in another device, type the new device name at the prompt, and press **Return**.

Performing a Simple Restoration

To perform a simple restoration of a database, use the **-create_database** (**-c**) option. For example, the following command restores the backup file, *employee.-gbk*, to a database named *employee.gdb*:

```
gbak -create employee.gbk employee.gdb
```

Unless you specify multiple target files, **gbak** restores the source as a single-file database. Typically, a restored database occupies less disk space than it did just before being backed up, but disk space requirements could change if the on-disk structure (ODS) version changes. For information about ODS, see "Upgrading to a New On-disk Structure," in this chapter.

The **-create** option prohibits overwriting a target file, so the previous command fails if *employee.gdb* already exists. To force **gbak** to overwrite an existing target database, use the **-replace_database** (**-r**) option:

```
gbak -r employee.gbk employee.gdb
```

Caution Using **-replace** is discouraged. When restoring to an existing file, a safer approach is to rename the existing database file, use **-create** to restore the backup file, then delete or archive the old database file as needed.

Restoring from a Storage Device

When restoring multiple tapes (or floppy disks) from a storage device, **gbak** pauses and displays the following prompt when it is ready to read the next tape:

```
Done with volume # number, "device"
   Press return to reopen that file, or type a new name
   followed by return to open a different file.
Name: default_device
```

The actual volume number and default device name will appear. To continue restoring from the same device, insert a new tape and press **Return**. To continue restoring from another device, insert a tape in another device, type the new device name at the prompt, and press **Return**.

Monitoring the Status of Backup and Restore

Verifying what **gbak** is doing as it runs can be helpful. The **-verify** (**-v**) option displays status messages on your screen. For example,

```
> gbak -backup -verify emp.gdb emp.gbk

   gbak: readied database emp.gdb for backup
   gbak: creating file emp.gbk
   gbak: starting transaction
   gbak: database emp.gdb has a page size of 2048 bytes.
   gbak: writing global fields
   gbak: writing global field ADDRESS
   gbak: writing global field AMOUNT
   gbak: writing global field BUDGET_SUBS
. . .
```

Instead of displaying messages on the screen, you can redirect them to an output file by using the **-y** option. Specify a destination path name as an argument to **-y**. The following command redirects the output of the previous example to the file, *messages_emp*:

```
/interbase> gbak -b -v -y messages_emp emp.gdb emp.gbk
```

You can suppress status messages by following **-y** with suppress_output instead of a file name. For example:

```
/interbas> gbak -b -v -y suppress_output emp.gdb emp.gbk
```

Making a Transportable Backup

To move a database to another machine that is not on the network, use the **-transportable** (**-t**) option. The **-transportable** option writes data in a generic format, so you can restore to any machine that supports InterBase.

To make a transportable backup:

1. Back up the database using the **-transportable** (**-t**) option. For example, if backing up to a file, enter:

   ```
   gbak -backup -transport employee.gdb employ_t.gbk
   ```

   If backing up to a removable medium, such as tape, specify the device name instead of a backup file:

   ```
   gbak -backup -transport employee.gdb /dev/rst0
   ```

2. If you backed up to a removable medium, proceed to Step 3. If you created a backup file, use operating-system commands to copy the file to tape, then load the contents of the tape onto another machine.
3. On the destination machine, restore the backup file. For example, if restoring from a file, enter:

   ```
   gbak -create employ_t.gbk employee.gdb
   ```

   If restoring from a removable medium, such as tape, specify the device name instead of the backup file:

   ```
   gbak -create /dev/rst0 employee.gdb
   ```

Tip If you work in a heterogeneous environment, you should make transportable backups regularly, even if the database is on a networked machine. If the network connection is interrupted, a remote database will not be available unless a transportable copy is placed on local machines.

Backup Options

Backup options include:

- Backing up metadata only
- Preventing garbage collection
- Ignoring checksums
- Ignoring limbo transactions

Backing Up Metadata

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 will have the same metadata but can be populated with different data.

Use the **-meta_data** (**-m**) option to back up metadata only. For example, back up the metadata of database *emp.gdb* as follows:

```
gbak -backup -meta emp.gdb emp.gbk
```

To create a new database using the same metadata as *emp.gdb*, restore *emp.gbk* to a new file, which will contain no data:

```
gbak -create emp.gbk new_emp.gdb
```

You can also extract a database's metadata using **isql -a**. This produces a data definition file (a text file), whereas **gbak -meta** creates a backup file (containing metadata only).

Preventing Garbage Collection

Normally during a backup, **gbak** performs garbage collection. Garbage collection physically erases old versions of records from disk. The **-garbage_collect** (**-g**) option prevents garbage collection during a backup. This option is rarely used. It might be useful if there is data corruption in old record versions and you want to prevent InterBase from visiting those records during a backup.

Ignoring Checksums

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, and **gbak** normally prevents you from backing up a database if bad checksums are detected. To ignore checksums during a backup, use the **-ignore** option. Be sure to examine the data the next time you restore the database.

Ignoring Limbo Transactions

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, you typically use **gfix** to perform automated recovery in case some transactions need to be committed.

If you know that limbo transactions can be safely rolled back, you do not need to perform automated transaction recovery. Instead, use the **-limbo** (**-l**) option. For example,

```
gbak -backup -limbo my.gdb my.gbk
```

When you back up using **-limbo**, **gbak** ignores all record versions created by any limbo transaction, finds the most recently committed version of a record, and backs up that version.

Restoration Options

Restoration options include:

- Splitting a database into more than one file
- Making indexes inactive
- Disabling validity checking
- Changing database page size
- Restoring a database without its shadow
- Restoring data incrementally

Splitting a Database into Several Files

By default, when **gbak** restores a backup, it creates a single database file. You can override this default and restore to multiple files. You can then distribute a database among different disks, which gives you more flexibility in allocating system resources.

To create a multi-file database from a backup file, use the following syntax:

```
gbak -create backup primary m secondary1 [n1 secondary2 [n2] ...]
```

Table 6-8: Backup Options

Argument	Description
backup	The *.gbk* file containing the database backup.
primary	The primary database file. It contains the metadata and is the first file used by InterBase. The remaining database pages are allocated to one or more secondary files.
m	Desired length of *primary* in database pages; minimum value is 200 pages.
secondary1	First of the additional (secondary) files in a multi-file database.
n1	Desired length of *secondary1* in database pages; no minimum value. If you specify only one secondary file, *n1* is optional because the last file will be as large as necessary to store the rest of the database, regardless of assigned page length.
secondary2	The next secondary file in a multi-file database. You can specify as many secondary files as needed.
n2	Desired length of *secondary2* in database pages; no minimum value. Specifying the length of the last secondary file is not required.

For example, to split database *big.gdb* into multiple files, first back it up:

```
gbak -verify -backup big.gdb big.gbk
```

Then restore the backup file into multiple database files:

```
gbak -verify -create big.gbk big0.gdb 1000 /x/big1.gdb 500 /y/big2.gdb
```

The length of the primary file, *big0.gdb*, is specified as 1,000 pages. If a value less than 200 is given, InterBase automatically increases it to 200 pages.

Note In the example, directories /*x* and /*y* represent different disks on the machine. It is good practice to store secondary files on separate disks, because their purpose is to let databases grow beyond the limits of a single disk. You must also ensure that all files in a database can be accessed directly by whatever program you run.

Making Indexes Inactive

Normally, **gbak** rebuilds indexes when a database is restored. If the database contained duplicate values in a unique index when it was backed up, **gbak** will fail when you attempt to restore the database.

To override this, use the **-inactive** (**-i**) option when you restore. This makes indexes inactive and disables rebuilding. For example,

```
gbak -create -inactive employee.gbk employee.gdb
```

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).

If **gbak** fails because of duplicate values, restore the database using **-inactive**, fix the duplicate values, and activate indexes using the SQL statements CREATE INDEX or ALTER INDEX. A unique index cannot be activated using the ALTER INDEX statement; a unique index must be dropped and then created again. For more information on activating indexes, see the *Language Reference*.

Note The **-inactive** option is also useful for bringing the database online more quickly. Data access will be slower until the indexes are rebuilt, but at least the database is available. After the database is restored, users can access the database while you reactivate the indexes.

Disabling Validity Checking

If you redefine validity constraints in a database where data is already entered, your data may no longer satisfy the validity constraints. You might not discover this until you try to restore the database, at which time **gbak** generates an invalid data error.

Caution Always make a copy of metadata before redefining it (for example, by extracting it using **isql**).

To restore a database that contains invalid data, use the **-no_validity** (**-no**) option:

```
gbak -create -no employee.gbk employee.gdb
```

The **-no_validity** 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.

The **-no_validity** 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.

Changing the Database Page Size

InterBase supports database page sizes of 1024, 2048, 4096, and 8192 bytes. The default is 1024 bytes. Use the **-page_size** (**-p**) option to change the page size.

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 BLOB data exceeding 1K, using a larger page size reduces the time for accessing the BLOB.
- 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 hierarchy. Indexes work faster if their depth is kept to a minimum. Consider increasing the page size if index depth is greater than 2 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.

To change the page size of a database, back up the database as you normally would. Then restore the database, specifying **-page_size** and a new page size (in bytes). The following example changes the page size of *emp.gd*B to 2048 bytes:

```
gbak -backup emp.gdb emp.gbk
gbak -create -page 2048 emp.gbk emp_new.gdb
```

Restoring a Database Without Its Shadow

Use the **-kill** option to restore a database without restoring its shadow. You might want to do this to restore a database created on a platform that does not support shadowing. When a database is restored using **-kill**, the definition of its shadow is deleted in the restored database.

Restoring Data Incrementally

Normally, **gbak** restores all metadata before restoring any data. If you specify the **-one_at_a_time** (**-o**) option, **gbak** restores the metadata and data for each table, committing one table at a time. The **-o** 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; **gbak** fails the moment it encounters bad data.

Upgrading to a New On-disk Structure

New major releases of InterBase 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.

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

1. Before installing the new version of InterBase, back up your databases using the old version of **gbak.** For confirmation, display the **gbak** version and ODS version by supplying the **-z** option. For example, if *emp.gdb* is a database created with InterBase 3.3, back it up as follows:

   ```
   gbak -z -verify -backup emp.gdb v33_emp.gbk
   ```

2. After the new version of InterBase is installed, use the new **gbak** to restore the backup files you created. As before, use **-z** to confirm the **gbak** and ODS versions. For example, after creating *v33_emp.gbk*, restore it as follows:

   ```
   gbak -z -v -c v33_emp.gbk v4_emp.gdb
   ```

CHAPTER 7

Shadowing a Database

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.

Overview of 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," in this chapter.

- 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 DBA, depending on how the shadow was defined. For more information about activating a shadow, see "Activating a Shadow," in this chapter.

- 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,” in this chapter.

- 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,” in this chapter.

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 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 disk. 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. In addition, it is advisable for a shadow to reside on a different node from the database.

- 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 DBA creates a shadow either in auto mode or in manual mode. For more information about these modes, see "Creating a Shadow in Auto Mode or Manual Mode," in this chapter.

- Automatic shadow creation

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

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

- Single-file or multi-file 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/dave/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/employee.shd" auto
   PAGE_SIZE 1024
   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 Multi-file Shadow

If the size of a database exceeds the space available on one disk, create a multi-file shadow and spread the files over several disks. To create a multi-file shadow, specify the name and size of each file in the shadow set. For example,

```
SQL> CREATE SHADOW 1 "employee.shd" LENGTH 1000
CON> FILE "emp1.shd" LENGTH 2000
CON> FILE "emp2.shd" LENGTH 2000;
```

The previous example creates a shadow set consisting of three files. The primary file, *employee.shd*, is 1,000 database pages in length. The secondary files, identified by the FILE keyword, are each 2,000 database pages long.

Instead of specifying the page length of secondary files, you can specify their starting page. The previous example could be entered as follows:

```
SQL> CREATE SHADOW 1 "employee.shd" LENGTH 1000
CON> FILE "emp1.shd" STARTING AT 1000
CON> FILE "emp2.shd" STARTING AT 3000;
```

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/employee.shd" auto length 1000
    file /usr/interbase/emp1.shd length 2000 starting 1000
    file /usr/interbase/emp2.shd length 2000 starting 3000
   PAGE_SIZE 1024
```

```
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.

Creating a Shadow in Auto Mode or 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 DBA intervenes. The benefits of auto mode and manual mode are compared in the following table:

Table 7-1: Auto vs. Manual Shadows

Mode	Advantage	Disadvantage
Auto	Database operation is uninterrupted.	Creates a temporary period when the database is not shadowed. The DBA 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 DBA.

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 may 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 "Creating a Conditional Shadow," in this chapter.

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, 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.

Creating a Conditional Shadow

A shadow can be defined so that if it replaces a database, a new shadow will be automatically created, 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, 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 might want to 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.

Caution DROP SHADOW deletes shadow references from a database's metadata, as well as the physical files on disk.

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 multi-file 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.

Appendix A

Error Messages

This appendix presents a list of error messages generated by the InterBase utilities for database administration. Along with the text of each error message is a suggested action to correct the problem.

The list of error messages is organized by utility:

- Backup and Restore (**gbak**)
- Database Maintenance (**gfix**)
- Security (**gsec**)

Error Messages for Backup and Restore

Table A-1: Error Messages for Backup and Restore

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 the **-factor** option.
Cannot commit files	Database may contain corruption, or metadata may violate integrity constraints. Try restoring tables using the **-one_at_a_time** option, or delete validity constraints using the **-no_validity** option.
Cannot commit index <*string*>	Data may conflict with defined indexes. Try restoring using the **-inactive** option to prevent rebuilding indexes.
Cannot find column for BLOB	
Cannot find table <*string*>	

Table A-1: Error Messages for Backup and Restore (Continued)

Error Message	**Causes and Suggested Actions to Take**
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 re-execute.
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.
Could not write to file <*string*>	Fix the file name and re-execute command.
Data type *n* not understood	An illegal data type 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. It may be necessary to back up the database using the **-expand** or **-old** options before it can be restored.
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**.
Device type <*string*> not known	The **-device** option (Apollo only) was used incorrectly.
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 may be corrupt. If restoring a database, try using **-one_at_a_time** to isolate the table.

Table A-1: Error Messages for Backup and Restore (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*	The problem array may need to be redefined.
Expected array version number *n* but instead found *m*	The problem array may need to be redefined.
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 may 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.
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.

Table A-1: Error Messages for Backup and Restore (Continued)

Error Message	**Causes and Suggested Actions to Take**
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 unassigned to a 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	
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.
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	

Table A-1: Error Messages for Backup and Restore (Continued)

Error Message	**Causes and Suggested Actions to Take**
Unexpected end of file on backup file	Restoration of the backup file failed. The backup procedure that created the backup file may 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 may 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. It may be necessary to delete 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*	

Error Messages for Database Maintenance

Table A-2: Messages for Database Maintenance

Error Message	**Causes and Suggested Actions to Take**
Invalid switch	A command-line option was not recognized.
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.
Please retry, specifying <*string*>	Both a file name and at least one option must be specified.
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.
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).
Numeric value required	The **-housekeeping** option requires a single, non-negative argument specifying number of transactions per sweep.
Transaction number or “all” required	You specified **-commit**, **-rollback**, or **-two_phase** without supplying the required argument.

Error Messages for Security Utility

Table A-3: Error Messages for Security Utility

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**.
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.

Appendix B

isql Command Reference

This chapter describes the syntax and usage for commands available only in InterBase **isql** (interactive SQL). For a description of the standard SQL commands available in **isql**, see the *Language Reference*.

Command-line **isql** supports the following special commands:

Table B-1: **isql** Commands

BLOBDUMP	SET BLOBDISPLAY	SHELL	SHOW INDEX
EDIT	SET COUNT	SHOW CHECK	SHOW PROCEDURES
EXIT	SET ECHO	SHOW DATABASE	SHOW SYSTEM
HELP	SET LIST	SHOW DOMAINS	SHOW TABLES
INPUT	SET NAMES	SHOW EXCEPTIONS	SHOW TRIGGERS
OUTPUT	SET PLAN	SHOW FILTERS	SHOW VERSION
QUIT	SET STATS	SHOW FUNCTIONS	SHOW VIEWS
SET	SET TERM	SHOW GENERATORS	
SET AUTODDL	SET TIME	SHOW GRANT	

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 (:). The first number is the ID of the table containing the BLOB column. The 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.

EDIT calls the text editor specified by the EDITOR environment variable. If this environment variable is not defined, then it uses the Unix text 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.

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 the following statement interactively:

```
SELECT DISTINCT JOB_CODE, JOB_TITLE FROM JOB;
```

Instead, the user mistakenly omits the DISTINCT keyword, so the statement is edited and re-executed:

```
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.

Caution 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, **isql** commands, or both.

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 non-interactive 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.

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 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 **-output** *filename* from the command line.

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.

Caution 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, ROLLBACK

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 a value.

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.

The following table lists supported SET commands:

Table B-2: SET Commands

SET	SET AUTODDL
SET BLOBDISPLAY	SET COUNT
SET ECHO	SET LIST
SET NAMES	SET PLAN
SET STATS	SET TERMINATOR
SET TIME	

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:       NONE
   Terminator:              ;
```

The output shows that **isql** will not echo commands, will display BLOB data if they are of subtype 1, will automatically commit DDL statements, and will 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 TERMINATOR, SET TIME

SET AUTODDL

Determines 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 commit of DDL (default).
OFF	Turns off automatic commit of DDL.

Description SET AUTODDL is used to turn on or off the automatic commit of DDL statements. By default, DDL statements are automatically committed after they are executed, in a separate transaction. This is the recommended behavior.

If the OFF keyword is specified, autocommit of DDL is turned off. Then, DDL statements can only be committed explicitly through a user's transaction. This may be useful for database prototyping, because uncommitted changes are easily undone by rolling them back.

Using SET AUTODDL OFF from within an **isql** session has the same effect as using **-noauto** from the command line. Using SET AUTODDL ON restores the default behavior.

Note The ON and OFF keywords are optional. If they are omitted, the command toggles automatic commit from ON to OFF or OFF to ON.

Examples The following example turns AUTODDL on, showing status before and after:

```
SET;
   No echo
   set blob 1
   Terminator: ;
SET AUTO ON;

SET;
   No echo
   set blob 1
   autocommit DDL
   Terminator: ;
```

See Also EXIT, QUIT, SET

SET BLOBDISPLAY

Determines whether to suppress the display of data associated with a BLOB ID.

Syntax `SET BLOBDISPLAY [n | ALL | OFF];`

Argument	Description
n	Integer specifying the BLOB subtype to display. Use 0 for BLOB data of unknown subtype; use 1 (default) for BLOB data of text subtype.
ALL	Display BLOB data of all subtypes.
OFF	Turn 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.

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. The output shows only the BLOB ID.

```
SET BLOB OFF;
SELECT PROJ_NAME, PROJ_DESC FROM PROJECT;

   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 BLOB display to subtype 1 (text). The contents of the BLOB appear below the corresponding BLOB ID.

```
SET;
   No echo
   autocommit DDL
   Terminator: ;
SET BLOB 1;
SET;
   No echo
   set blob 1
   autocommit DDL
   Terminator: ;
SELECT PROJ_NAME, PROJ_DESC FROM PROJECT;

   PROJ_NAME                PROJ_DESC
```

```
====================     =================
Video Database           24:6
==============================================================
PROJ_DESC:
Design a video data base management system for
controlling on-demand video distribution.

==============================================================
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

SET COUNT

Determines whether to display a message indicating the number of rows retrieved.

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.

Note The ON and OFF keywords are optional. If they are omitted, the command toggles row count from ON to OFF or OFF to ON.

Example To display the number of rows returned by any query, enter:

```
SET COUNT ON;
SELECT * FROM COUNTRY
   WHERE CURRENCY LIKE "%FRANC%";
```

```
COUNTRY                  CURRENCY
===============          ==========

SWITZERLAND              SFRANC
FRANCE                   FFRANC
BELGIUM                  BFRANC

3 rows returned
```

See Also SET

SET ECHO

Determines whether commands are redisplayed before being executed.

Syntax

```
SET ECHO [ON | OFF];
```

Argument	Description
ON	Turns on command echoing.
OFF	Turns off command echoing (default).

Description By default, commands are not redisplayed, or echoed, before being executed.

Use SET ECHO ON to change the default behavior and echo commands. Command echoing is useful when commands are read from an input file, because normally you do not see commands that are executed from a file. Command echoing also enables a command to be recorded in an output file, along with the data retrieved.

Using SET ECHO ON from within an **isql** session has the same effect as using **-echo** from the command line. To turn command echoing off, use SET ECHO OFF. This restores the default behavior.

Note The ON and OFF keywords are optional. If they are omitted, the command toggles echo from ON to OFF or OFF to ON.

Example To turn on command echoing, enter:

```
SET ECHO ON;
```

See Also INPUT, OUTPUT, SET

SET LIST

Determines 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.

Note The ON and OFF keywords are optional. If they are omitted, the command toggles list format display from ON to OFF or OFF to ON.

Example The following example shows output from a SELECT statement that appears in list format. The salary values are in lira.

```
SET LIST ON;
SELECT * FROM JOB
   WHERE JOB_COUNTRY = "Italy";

   JOB_CODE                 SRep
   JOB_GRADE                4
   JOB_COUNTRY              Italy
   JOB_TITLE                Sales Representative
   MIN_SALARY               33600000.00
   MAX_SALARY               168000000.00
   JOB_REQUIREMENT          <null>
   LANGUAGE_REQ             <null>
```

See Also SET

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: NONE.

Description SET NAMES specifies the character set to use for subsequent database attachments 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 the 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 To change the active character set to ISO8859_1, enter:

```
SET NAMES ISO8859_1;
CONNECT employee.gdb;
```

See Also SET

SET PLAN

Determines 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.

Note The ON and OFF keywords are optional. If they are omitted, the command toggles display of query plan from ON to OFF or OFF to ON.

Example The following example shows output from a SELECT statement as well as the query optimizer plan that was used to retrieve the data:

```
SET PLAN ON;
SELECT JOB_COUNTRY, MIN_SALARY FROM JOB
   WHERE MIN_SALARY > 50000
      AND JOB_COUNTRY = "France";

   PLAN (JOB INDEX (RDB$FOREIGN3,MINSALX,MAXSALX))

   JOB_COUNTRY              MIN_SALARY
   ===============          ======================
   France                   118200.00
```

See Also SET

SET STATS

Determines 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 available memory, in bytes

- Change in available memory, in bytes (useful when adding or deleting data)
- Maximum available memory, 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.

Do not confuse SET STATS with the SQL statement SET STATISTICS, which recalculates the selectivity of an index.

Note The ON and OFF keywords are optional. If they are omitted, the command toggles statistics display from ON to OFF or OFF to ON.

Example The following example shows output from a SELECT statement as well as the performance statistics for the operation:

```
SET STATS ON;
SELECT JOB_COUNTRY, MIN_SALARY FROM JOB
   WHERE MIN_SALARY > 50000
      AND JOB_COUNTRY = "France";

   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

Determines 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: semicolon (;).

Description By default, **isql** commands must be terminated by a semicolon (;). Use SET TERM to change the terminator character.

SET TERM is typically used with CREATE PROCEDURE or CREATE TRIGGER. Procedures and triggers are defined using procedure and trigger language in which statements always end with a semicolon. The procedure or trigger itself must then be terminated by a character other than a semicolon.

A text 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.

Using SET TERM from within an **isql** session has the same effect as using **-terminator** from the command line.

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

Determines 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 data type 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.

Note 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 turns the time display on:

```
SET TIME ON;
```

See Also SET

SHELL

Allows execution of an operating system command or temporary access to an operating system shell.

Syntax

```
SHELL [<os_command>];
```

Argument	Description
<os_command>	A Unix command. If no command is specified, **isql** provides interactive access to the operating system.

Description The SHELL command provides temporary access to Unix commands in an **isql** session. Use SHELL to execute an operating system command without ending the current **isql** session.

If <*os_command*> is specified, Unix executes the command and then returns to **isql** when complete.

If no command is specified, a Unix 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.

Example The following example uses SHELL to display the contents of the current directory:

```
SHELL ls -l;
```

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 TABLES.

Example The following example shows CHECK constraints defined for the table, JOB. 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;
```

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:

```
isql
   Use CONNECT or CREATE DATABASE to specify a database
CONNECT "employee.gdb";
   Database: employee.gdb
SHOW DB;
   Database: employee.gdb
         Owner: SYSDBA
   PAGE_SIZE 1024
   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.

Description 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 data type, 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 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
      Invalid employee number or project id.
. . .
```

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.

Description 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; namely, the input subtype, output subtype, module (or library) name, and entry point.

Example The following lists all filters and then shows the definition of 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.

Description 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 data types of return values and input arguments.

Example The following example lists all UDFs and shows the definition of MAXNUM:

```
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.

To create a generator, use CREATE GENERATOR. To assign a starting value, use SET GENERATOR. To insert a generated value into a database, use GEN_ID().

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, REFERENCES, SELECT, UPDATE, or ALL. To change privileges, use the SQL statements GRANT or REVOKE.

Before using SHOW GRANT, you can list the available database objects. Use SHOW PROCEDURES, SHOW TABLES, and SHOW VIEWS.

Example To display GRANT privileges on the table JOB enter:

```
SHOW GRANT JOB;
  GRANT SELECT ON JOB TO ALL
  GRANT DELETE, INSERT, SELECT, UPDATE ON JOB TO MANAGER
```

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 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 TABLE. If no argument is specified, SHOW INDEX displays information for all indexes in the current database. SHOW INDEX has a shorthand equivalent, SHOW IND.

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_ID Exception
   DELETE_EMPLOYEE          DEPARTMENT     Table
                            EMPLOYEE       Table
                            EMPLOYEE_PROJECTTable
                            PROJECT        Table
                            REASSIGN_SALES Exception
                            SALARY_HISTORY Table
                            SALES          Table
   DEPT_BUDGET              DEPARTMENT     Table
                            DEPT_BUDGET    Procedure
. . .
```

To display the text of 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 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$SECURITY_CLASSES                  RDB$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 and FOREIGN KEY, 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 list 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 VIEW, 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.

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).

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: HP-B4.0C
   InterBase/UNIX (access method), version "HP-B4.0C"
   on disk structure version 8.0
```

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 TABLE

Appendix C

Command-line DBA Utilities Reference

This chapter is a reference for the command-line DBA utilities, **gsec**, **gfix**, and **gbak**. For a general discussion of using these utilities, see Chapter 6: "Using the Command-line DBA Utilities."

gbak

Backs up or restores a database, optionally changing database characteristics.

Syntax For backing up:

```
gbak [-b] [options] database target
```

For restoring:

```
gbak {-c|-r} [options] source database
```

For restoring to multiple files:

```
gbak {-c|-r} [options] source primary m secondary1 [n1 secondary2 [n2]]
```

Table C-1: **gbak** Argument Descriptions

Argument	Description
options	See "Options" below.
database	Name of a database to back up or restore.
source	Name of a storage device or backup file from which to restore.
target	Name of a storage device or backup file to which to back up.
primary	Primary file when restoring to multiple database files.
m	Length of *primary* in database pages; minimum value is 200.
secondary1	First secondary file when restoring to multiple database files.

Table C-1: **gbak** Argument Descriptions (Continued)

Argument	Description
n1	Length of *secondary1*; if only one secondary file is supplied, *n1* is optional.
secondary2	Next secondary file; specify as many secondary files as needed.
n2	Length of *secondary2*; it is unnecessary to specify the length of the last secondary file.

Options

In the Option column of the following tables, only the characters outside the brackets ([ ]) are required.

Table C-2: **gbak** Backup Options

Option	Description
-b[**ackup_database**]	Back up database to file or device.
-e[**xpand**]	Do not create a compressed back up.
-fa[**ctor**] *n*	Use blocking factor *n* for tape device.
-g[**arbage_collect**]	Do not garbage collect during backup.
-ig[**nore**]	Ignore checksums during backup.
-l[**imbo**]	Ignore limbo transactions during backup.
-m[**eta_data**]	Back up metadata only, no data.
-ol[**d_descriptions**]	Back up metadata in old-style format.
-pa[**ssword**] *text*	Check for password *text* before accessing a database.
-t[**ransportable**]	Create a transportable back up (useful for copying a database to different operating systems).
-u[**ser**] *name*	Check for user *name* before accessing remote database.
-v[**erify**]	Show what **gbak** is doing.
-y [***file* | suppress_output**]	Direct status messages to *file* or suppress output messages. Suppresses messages if *file* is omitted.
-z	Show version of **gbak** and of InterBase engine.

Table C-3: **gbak** Restore Options

Option	Description
-c[**reate_database**]	Restore database to a new file.
-i[**nactive**]	Make indexes inactive upon restore.
-k[**ill**]	Do not create any shadows that were previously defined.
-n[**o_validity**]	Delete validity constraints from restored metadata; allows restoration of data that would otherwise not meet validity constraints.

Table C-3: **gbak** Restore Options (Continued)

Option	Description
-o[**ne_at_a_time**]	Restore one table at a time; useful for partial recovery if database contains corrupt data.
-p[**age_size**] *n*	Reset page size to *n* bytes (1024, 2048, 4196, or 8192). Default: 1024.
-pa[**ssword**] *text*	Check for password *text* before accessing a database.
-r[**eplace_database**]	Restore database to new file or replace existing file.
-u[**ser**] *name*	Check for user *name* before accessing remote database. Required when using **gbak** from a Windows Client.
-v[**erify**]	Show what **gbak** is doing.
-y [***file*** **| suppress_output**]	If used with **-v**, direct status messages to *file;* if used without **-v** and *file* is omitted, suppress output messages.
-z	Show version of **gbak** and of InterBase engine.

Description

gbak performs database backup and restoration. In addition to simple backups and restores, **gbak** can be used to upgrade the on-disk structure (ODS), copy a database from one machine type to another, split a database into multiple files, or change the database page size.

Because **gbak** does not require exclusive access to a database, online backups are allowed. Regular backups and restores with **gbak** improve database performance by balancing indexes, by packing data more efficiently, and by grouping tables in a single area of the database file.

Examples

Back up database *employee.gdb* to file *employee.gbk*, displaying status messages:

```
gbak -backup -verify employee.gdb employee.gbk
```

Restore *employee.gbk*, splitting the created database among three files; *newemp.gdb* is 1,000 pages long, *emp1.gdb* is 500 pages long, and *emp2.gdb* is as large as necessary to store the rest of the database:

```
gbak -create employee.gbk newemp.gdb 1000 emp1.gdb 500 emp2.gdb
```

Change the page size of *newemp.gdb* to 2048 bytes:

```
gbak -backup newemp.gdb newemp.gbk
gbak -create -page 2048 newemp.gbk employee.gdb
```

Restore database *stocks.gdb* without its shadow:

```
gbak -create -kill stocks.gbk stocks.gdb
```

Back up a database that resides on a remote server named *hera*. Note that a user name and password are needed to access the remote database:

```
gbak -b -user "burt" -pass "codeword" hera:/usr/burt.gdb my.gbk
```

Back up a database to a cartridge tape. The tape is loaded in a device named */dev/rst0*:

```
gbak -b employee.gdb/dev/rst0
```

gfix

Performs various maintenance activities on a database.

Syntax

```
gfix [options] db_name
```

Options

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

Table C-4: **gfix** Options

Option	Usage	Description
-at[**tach**] *n*	Shutdown	Used with **-shut** to prevent new database attachments after *n* seconds.
-ca[**che**]	Shutdown	Reserved for future releases of InterBase.
-c[**ommit**] {*id* | **all**}	Transaction recovery	Commit limbo transaction specified by *id* or commit all limbo transactions.
-f[**orce**] *n*	Shutdown	Used with **-shut** to force immediate shutdown of a database, beginning in *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	Change automatic sweep interval to *n* transactions, or disable sweeping by setting *n* to 0. Default interval is 20,000 transactions. Exclusive access is not needed.
-i[**gnore**]	Data repair	Ignore checksum errors when validating or sweeping.

Table C-4: **gfix** Options (Continued)

Option	Usage	Description
-l[**ist**]	Transaction recovery	Display IDs of each limbo transaction and indicate what would occur if **-t** were used for automated two-phase recovery.
-m[**end**]	Data repair	Mark corrupt records as unavailable, so they are skipped (for example, during a subsequent back up).
-n[**o_update**]	Data repair	Used with **-v** to validate corrupt or mis-allocated structures. Structures are reported but not fixed.
-o[**nline**]	Shutdown	Cancels a **-shut** operation that is scheduled to take effect or that is currently in effect.
-pa[**ssword**] *text*	Remote access	Check for password *text* before accessing a data-base.
-p[**rompt**]	Transaction recovery	Used with **-l** to prompt for action during transaction recovery.
-r[**ollback**] {*id* | **all**}	Transaction recovery	Roll back limbo transaction specified by *id* or roll back all limbo transactions.
-s[**weep**]	Sweeping	Force an immediate sweep of the database. Useful if automatic sweeping is disabled. Exclusive access is not necessary.
-sh[**ut**]	Shutdown	Shut down the database. Must also specify either **-attach**, **-force**, or **-tran**.
-t[**wo_phase**] {*id* | **all**}	Transaction recovery	Perform 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.
-user *name*	Remote access	Check for user *name* before accessing a remote database.
-v[**alidate**]	Data repair	Locate and release pages that are allocated but unassigned to any data structures. Also reports cor-rupt structures.
-w[**rite**]		Enable and disable forced (synchronous) writes.
-z		Show version of **gfix** and of InterBase engine.

Description

gfix can be used to shut down the database, make minor repairs to data struc-tures, and recover transactions.

The **gfix** options expect a database as the *file* argument.

Examples

Schedule *employee.gdb* to deny further attachments in an hour:

```
gfix -shut -attach 3600 employee.gdb
```

Disable automatic sweeping for *stocks.gdb*. This task is typically a prelude to setting up database sweeps at scheduled times:

```
gfix -housekeeping 0 stocks.gdb
```

Perform automated two-phase recovery on *sales.gdb*:

```
gfix -list -two_phase sales.gdb
```

gsec

Display, add, modify, or delete information in the security database.

Syntax

```
gsec [options] [arguments]
```

Options

In the Option column of the following table, only the characters outside the brackets ([ ]) are required. You may specify any additional characters up to and including the full option name.

Table C-5: **gsec** Options

Option	Description
-a[**dd**] *name*	Add user's login name to security database. Supply additional *arguments* to specify user information.
-de[**lete**] *name*	Remove row containing user *name* from security database.
-di[**splay**]	Display all rows of security database.
-di[**splay**] *name*	Display security information for user *name*.
-h[**elp**]	Display syntax and usage of **gsec** commands.
-m[**odify**] *name*	Change information for user's login *name* in security database. Supply additional arguments to specify user information.
-z	Show version of **gsec** and of InterBase engine.

In interactive mode, use the **gsec** options as commands by omitting the leading hyphen. Additional interactive commands include **?** (same as **help**) and **quit**, which ends the interactive session.

Arguments

Arguments correspond to columns in the *isc4.gdb* security database and are specified only when using **-add** or **-modify**. Arguments must always be specified with a leading hyphen, even if **gsec** is used in interactive mode. No arguments are required, but specify at least a password when adding new users.

Table C-6: **gsec** Argument Descriptions

Argument	Description
-f[**name**] *name*	User's first name
-g[**id**] *id*	User's group id (an integer)
-l[**name**] *name*	User's last name
-**mn**[**ame**] *name*	User's middle name
-o[**rg**] *text*	User's organization
-p[**roj**] *text*	User's project
-pw *text*	User's password
-u[**id**] *id*	User's user id (an integer)

Description

The **gsec** utility is used to manage database and server security. With no options or arguments on the command line, **gsec** enters interactive mode (marked by a GSEC> prompt). Otherwise, it performs the functions specified by the command-line options and arguments.

You must log in as root to use **gsec**.

Examples

The following command adds user IDAKNOW and assigns password **xyz123**:

```
gsec
GSEC> add IDAKNOW -pw xyz123
GSEC> quit
```

Index

A

activating shadows 74–75
adding
  shadow files 75
  user names 43
applications
  building 26
  improving performance 46, 66
assigning passwords 43
attachments, preventing 54
AUTO mode 73
automated transaction recovery 51, 52, 63
automatic commit of DDL statements 30, 89
automatic sweeping 46
  disabling 47

B

backing up databases 46, 48, 57
  previous InterBase versions 11
  remote 59
backup copies 45
backup files
  creating 58–60
    options 62
    transportable 61
  multiple 59
  naming 59
bad checksums 51
  ignoring 63
binary data 84
binary files 84
BLOB data
  displaying 90–92
  editing 83
  ID numbers 91
    retrieving 84
  saving 83
BLOB filters 27, 103
BLOBDUMP 38, 83
building applications 26

C

case, nomenclature 3
changing user names 44
character sets 95
CHECK constraints 100, 101
checksum errors 51
  ignoring 63
closing databases 85, 88
column headers 31
column names, nomenclature 3
command-line options
  gbak 57
  gfix
    shutdown 54
    sweeping 46
    transaction recovery 52
  gsec 43
  isql 30–31
    specifying 29–30
command-line utilities 41–67
  reference 111–117
commands
  gbak 111–114
  gfix 114–116
  gsec 42–44, 116–117
  isql 35–40
    displaying 85, 87, 93
    editing 84
    executing 86
    reference 83–110
COMMIT 32
commits 51
  automatic 30, 89
conditional shadows 74
CONNECT 32
connections
  databases 32
  remote servers 31, 32
constraints 100, 101
  disabling 65
  naming 4
continuation prompt 30
corrupt databases, fixing 48, 49–51
CREATE DATABASE 30, 32

creating
  backup files 58–60
  shadows 71–74
    conditional 74

D

data 29
  binary 84
  BLOB 83–84, 90–92
  losing 57
database administration 25, 46, 66
  adding users 43
  backing up databases 58–60, 61
    options 62
  changing sweep interval 46–47
  changing user names 44
  disabling sweep interval 47
  fixing corrupt databases 49
  maintenance utility 45–57
  recovering limbo transactions 51
  restarting databases 56
  restoring databases 60
    incrementally 67
    options 64
    to multiple files 64
  security utility 41–44
  shutting down databases 53–56
    canceling 57
  troubleshooting 48
  verifying data structures 48
database objects, naming 3
database pages 45
  fixing orphan pages 49
  resizing 66
database structures
  mending 49–51
  retrieving 101
  validating 48–49
databases
  backing up 46, 48, 57
  closing 85, 88
  fixing corrupt 48, 49–51
  garbage collection 45, 63
  maintaining 45–57
  repairing 47–51
    checksum errors 51
    limbo transactions 51–53
    mis-allocated structures 49–51
    orphan pages 49
  restarting 56
  shadowing 69–75
  shutting down 53–56
    canceling 57
  specifying character sets 95
  sweeping 45–47
DDL 35
  automatic commits 30, 89
  extracting 30, 31, 33–35
  processing 29
  transactions 32
DECLARE FILTER 103
default settings, isql 37
default sweep interval 46
default terminator 30
deleting
  objects 106
  records 45
  shadows 75
directories, path names 4, 5
disk crashes 57
displaying
  error messages 87
  isql commands 85, 87, 93
  metadata 35
  privileges 105
  statements 30
  version numbers 31, 110
DML 35
  processing 29
domains 101
DROP SHADOW 75

E

echoing 30, 93
EDIT 39, 84
  INPUT and 86
editing
  BLOB data 83
  input files 100
  isql commands 84

editors 84
environments, isql 37
error messages, displaying 87
errors
  checksum 51, 63
  corrupt database 48, 49–51
  isql 40
  system 49
exceptions 102
EXIT 85
  QUIT vs. 88
extracting
  DDL 30, 31
  metadata 33–35

F

file names
  backup copies 59
  nomenclature 4–5
files
  backup 59
  binary 84
  input 30, 86, 93, 100
  shadow 72, 75
  writing to 30, 33
filespec parameter 4
filters 103
formats, output 94

G

garbage collection 45
  preventing 63
gbak 57–67
  backing up databases 58–60, 61
    options 62
  commands 111–114
  preventing garbage collection 63
  rebuilding indexes 65
  resizing database pages 66
  restoring databases 60
    incrementally 67
    options 64
    to multiple files 64
  starting 58
  status messages 61
  syntax 58
  validity checking 65
generators 104
gfix 45–57
  activating shadows 74–75
  commands 114–116
  killing shadows 74
  options 46, 52
    shutdown 54
  repairing databases 47–51
    checksum errors 51
    limbo transactions 51–53
    mis-allocated structures 49–51
    orphan pages 49
  restarting databases 56
  shutting down databases 53–56
  sweeping 45–47
  syntax 45
gsec 41–44
  adding users 43–44
  changing entries 44
  commands 42–44, 116–117
    summarized 42
  deleting entries 44
  exiting 42
  options 43
  starting 42, 44

H

HELP 85
housekeeping 46, 47

I

ignoring bad checksums 63
improving application performance 46, 66
indexes 35
  rebuilding 65
  retrieving 105
INPUT 39, 86
input files 30, 86
  editing 100
  reading 93
installing InterBase 9–18
integrity constraints 100, 101
  disabling 65

naming 4
InterBase
architecture 45
installing 9–18
licensing 13–15
version numbers 31, 110
international character sets 95
IPCSEMAPHORE 9
IPCSHMEM 9
isc4.gdb 41, 42
viewing contents 42
iscinstall utility 13
messages 16–17
isql 29–40
commands 35–40
displaying 85, 87, 93
editing 84
executing 86
reference 83–110
summarized 36, 37
connecting to databases 32
default settings 37
errors 40
exiting 31, 40, 85, 88
options 30–31
specifying 29–30
setting environment 37
starting 29–30, 32
terminator characters 30, 31, 98

K

killing shadows 67, 74

L

licensing InterBase 13–15
limbo transactions 63
defined 52
recovering 51
list formats 94
lock header files 11
losing data 57

M

maintaining databases 45–57
MANUAL mode 74
messages 92
gbak 61
metadata 29
backing up 62
displaying 35
extracting 33–35
redefining 66
mis-allocated structures 49–51
multi-file specifications 5
multiple backup files 59

N

naming conventions 3–5
nesting INPUT commands 86
node names, shared disks vs. 32
nodes, naming 4, 5
nomenclature 3–5

O

objects, deleting 106
on-disk structures, upgrading 67
operating system shells 99
orphan database pages 49
outdated records, removing 45
OUTPUT 87
output 87, 93
BLOB data 90–92
formatting 94
metadata 35
redirecting 87
statements 30
user-defined functions 103
output files 30, 33

P

parameters, filespec 4
passwords 31, 41
assigning 43
path names 4, 5
performance statistics 96
platforms 4
power failure 57
preventing
attachments 54
garbage collection 63

    new transactions 55
primary file specifications 4
printing conventions (documentation) 2–3
privileges, displaying 105
procedures 35
    defining 98
    listing 106

Q

queries 95, 99
    counting rows retrieved 92
    performance statistics 96
    retrieving rows 94
    testing 29
QUIT 88
    EXIT vs. 85

R

rebuilding indexes 65
records, removing outdated 45
recovering transactions 51, 52, 63
remote backups 59
remote servers, connecting to 31, 32
removing
    outdated records 45
    shadows 67
repairing databases 47–51
    checksum errors 51
    limbo transactions 51–53
    mis-allocated structures 49–51
    orphan pages 49
restarting databases 56
restrictions, nomenclature 4
ROLLBACK 32
rolling back transactions 88

S

secondary file specifications 4, 5
security 41–44
    changing user names 44
    displaying privileges 105
SEM_UNDO structures 9
semaphore facility 9
SET 88
SET AUTODDL 89
SET BLOBDISPLAY 90
SET COUNT 92
SET ECHO 93
SET LIST 94
SET NAMES 95
SET PLAN 95, 99
SET STATS 96
SET TERM 98
SET TERMINATOR 30
SET TRANSACTION 32
shadowing databases 69–75
    advantages 70
    limitations 70
    overview 69–70
shadows
    activating 74–75
    adding files 75
    AUTO mode 73
    conditional 74
    creating 71–74
    dropping 75
    killing 74
    MANUAL mode 74
    removing 67
shared disks, node names vs. 32
shared memory facility 9
SHELL 39, 99
SHOW CHECK 100
SHOW DATABASE 72, 101
SHOW DOMAINS 101
SHOW EXCEPTIONS 102
SHOW FILTERS 103
SHOW FUNCTIONS 103
SHOW GENERATORS 104
SHOW GRANT 105
SHOW INDEX 105
SHOW PROCEDURES 106
SHOW SYSTEM 107
SHOW TABLES 108
SHOW TRIGGERS 109
SHOW VERSION 110
SHOW VIEWS 110
shutting down databases 53–56
    bringing back online 56
    canceling 57

preventing attachments 54
preventing new transactions 55
SQLCODE 40
starting
gbak 58
gsec 42, 44
isql 29–30, 32
statements
displaying 30
example, printing conventions 2
terminator characters 30, 31, 98
storage devices
backing up databases 59
restoring databases 60
sweeping databases 45–47
automatically 46
changing sweep interval 46–47
disabling sweep interval 47
syntax
file name specifications 4
statements, printing conventions 2
system editors 84
system errors 49
system shells 99
system tables 107
system views 107

T

table names, nomenclature 3
tables 35
constraints 100, 101
disabling 65
listing 108
tabular formats 94
tape backups 59
restoring databases 60
TCP/IP 32
terminator characters
default 30
isql 30, 31, 98
testing
installation 18
queries 29
text editors 84
text, saving to files 84
transactions
commit process 51
committing 32, 85
DDL 32
isql 32–33
limbo 52, 63
recovering 51
preventing new 55
rolling back 32, 88
transportable backups 61
triggers 35
defining 98
listing 109
troubleshooting 48

U

user names 31, 41
adding 43–44
changing 44
user permissions 29
user-defined functions 27
viewing 103

V

validating database structures 48–49
validity checking 65
version numbers, displaying 31, 110
views
listing 108, 110
naming 3

W

write errors 49