SQLBase

Database Administrator’s Guide

20-2121-1005

centura™

Trademarks

Copyright

SQLBase Database Administrator’s Guide
20-2121-1005
November 1997

Contents

Preface

This manual provides reference information about setting up and maintaining SQLBase databases.

This preface provides the following information:

- Who should read this manual.
- The organization of this manual.
- The documentation format.
- The notation conventions used in this manual.
- Related publications.

Audience

This manual is intended for:

- **Application Developers** building client applications that access databases using Centura frontend products like SQLTalk, Team Developer, and the SQL/API.
- **Database Administrators** performing day-to-day operation and maintenance of the database and configuring the database settings to customize the environment.
- **System administrators** maintaining and configuring the operating system, such as configuring the physical and logical devices on NetWare used to install SQLBase. This includes maintaining operating system access security.
- **Network Administrator** maintaining and configuring the network. This includes network access security, logical and physical network address assignment, and network resources allocations.

This manual assumes you have a working knowledge of relational databases and SQL.

Organization of this manual

This manual is organized into the chapters listed below. There is also an index, and a glossary of frequently used terms.

Chapter 1: RDBMS Concepts
This chapter provides an overview of a DBA's responsibilities and relational database management systems.

Chapter 2: Communications
This chapter summarizes your communication options and defines each platform's client and server executables.

Chapter 3: Configuration (sql.ini)
This chapter provides extensive detail for each configuration file (sql.ini) keyword.

Chapter 4: Databases
This chapter discusses database issues such as cache, temporary files and log files, partitioning, and auditing.

Chapter 5: DBA Interfaces
This chapter introduces you to the SQLBase client software components that you use to perform DBA functions, and details the server interface.

Chapter 6: DBA Operations
This chapter touches on all aspects of a DBA's daily operations.

Chapter 7: Security and Authorization
This chapter shows you how to implement security and authorization through database authority, table and view privileges, and server security.

Chapter 8: Backing Up and Restoring Databases
This chapter discusses SQLBase's options for backing up and restoring databases.

Chapter 9: Distributed Transactions
This chapter describes how to create distributed transactions.

Chapter 10: ODBC Support
This chapter describes SQLBase’s support for the Microsoft DataBase Connectivity (ODBC) standard.

Chapter 11: National Language Support
This chapter briefly describes how you can customize SQLBase for non-English languages.

Chapter 12: NetWare Directory Service Support
This chapter describes SQLBase’s support for the NetWare Directory Service (NDS) 4.x.

Chapter 13:Running SQLBase Server as an NT Service
This chapter describes running SQLBase Server as an application program or a Windows NT service program, and using SQLBase Server Monitor (SSM).

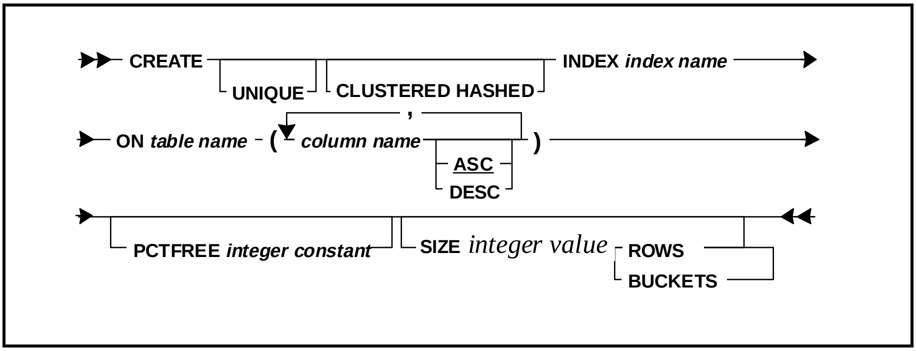
Appendix A
This appendix describes the contents of each table in the SQLBase system catalog.

Appendix B
This appendix describes SQLBase system procedures.

Appendix C
This appendix describes the SQLBase system utility tables.

Syntax diagrams

This manual uses syntax diagrams to show how to enter commands. The syntax for the CREATE INDEX command is used here as an example.



Read the syntax diagram from left to right and top to bottom.

The line with the command name (CREATE) is the main line of the command. Mandatory keywords and arguments (such as INDEX or ON ***table name***) appear on the main line or a continuation of the main line.

This example diagram could generate the commands shown in these examples:

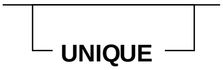
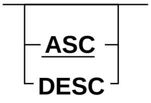
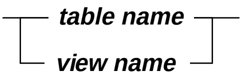
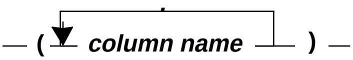
```
CREATE UNIQUE INDEX PARTIDX ON PARTS (PARTNO);

CREATE INDEX ORDER$DATE$IDX ON ORDERS (ORDERNUM, DATE);
```

Note that example statements in this manual can appear in bold to distinguish user entries from a system response:

```
ROWCOUNT EMP;

5 ROWS IN TABLE
```

The following table shows the syntax diagram symbols used in this manual.

Symbol	Description
►►	A double arrow pointing right means the start of a command.
──►	A single arrow pointing right means a continuation line of a command.
◄◄	The double arrow pointing left means the end of a command.

Symbol	Description
	Optional clauses and keywords (such as UNIQUE) hang off the main or continuation lines.
 	If there is an optional item with alternate choices, the choices are in a vertical list. In this example, ASC and DESC are alternate non-mandatory options. ASC is underlined, which means it is the default and can be omitted. If an item is mandatory, the first alternative is on the main line (this example is from the UPDATE command).
	When you can repeat arguments of the same type (such as a list of column names), an arrow pointing downward is suspended above the argument. A delimiter or operator on this line shows what separates each argument (such as commas separating column names).

Notation conventions

Before you start using this manual, it is important to understand the typographical conventions we use in this manual:

Formatting Convention	Type of Information
You	A developer who reads this manual
User	The end-user of applications that you write
bold type	Menu items, push buttons, and field names. Things that you select. Keyboard keys that you press.
Courier 9	Builder or C language code example
SQL.INI MAPDLL.EXE	Program names and file names
Precaution	**Warning:**
Vital information	**Important:**

Formatting Convention	Type of Information
Supplemental information	**Note:**
Alt+1	A plus sign between key names means to press and hold down the first key while you press the second key

Other helpful resources

Centura Books Online. The Centura document suite is available online. This document collection lets you perform full-text indexed searches across the entire document suite, navigate the table of contents using the expandable/collapsible browser, or print any chapter. Open the collection by selecting the Centura Books Online icon from the **Start** menu or by double-clicking on the launcher icon in the program group.

Online Help. This is an extensive context-sensitive online help system. The online help offers a quick way to find information on topics including menu items, functions, messages, and objects.

World Wide Web. Centura Software's World Wide Web site contains information about Centura Software Corporation's partners, products, sales, support, training, and users. The URL is http://www.centurasoft.com.

To access Centura technical services on the Web, go to http:/www.centurasoft.com/support. This section of our Web site is a valuable resource for customers with technical support issues, and addresses a variety of topics and services, including technical support case status, commonly asked questions, access to Centura's Online Newsgroups, links to Shareware tools, product bulletins, white papers, and downloadable product updates.

For information on training, including course descriptions, class schedules, and Certified Training Partners, go to http://www.centurasoft.com/training.

Send comments to...

Anyone reading this manual can contribute to it. If you have any comments or suggestions, please send them to:

Technical Publications Department
Centura Software Corporation
975 Island Drive
Redwood Shores, CA 94065

or send email, with comments or suggestions to:

techpubs@centurasoft.com

Chapter 1

RDBMS Concepts

This chapter briefly explains some concepts that you need to understand in order to use SQLBase software:

- A DBA's responsibilities.

  This sections discusses the responsibilities of a Database Administrator.

- Databases and networks

  This section describes client/server terminology as it applies to SQLBase.

Database administrator (DBA)

As a DBA, you ensure the smooth operation of one or more databases and you are responsible for the design, planning, installation, configuration, security, management, maintenance, and operation of a database management system (DBMS) and its supporting network.

A good way to ensure that the databases meet the needs of your organization is by becoming familiar with the organization's applications: who are its users, what data is stored and accessed, and what types of transactions occur.

A DBA's specific responsibilities include:

- Keeping the database servers running on a daily basis.
- Diagnosing and resolving system problems.
- Installing both database server and client software.
- Creating databases.
- Backing up and recovering databases.
- Controlling security and access to the database and its objects.
- Monitoring and tuning the performance of a database as well as the applications that access it.
- Managing communications.
- Managing disk space and estimating storage needs.
- Advising developers about table, index, and view design, multi-user considerations, networking, and loading, converting, and unloading data.
- Ensuring data availability, accuracy, completeness, integrity, and consistency.
- Auditing the use of a database.
- Administering the system catalog.
- Helping database application users.

Organizing the DBA function

There are two SQLBase authority levels that a DBA can have:

- SYSADM

  The most powerful authority level. There should only be one user with SYSADM authority. This user can designate one or more other users as DBAs.

- DBA

  A user with this authority level has privileges on all database objects and can grant, change, or revoke the object privileges of any user but SYSADM.

Access to the SYSADM and DBA accounts should be tightly controlled.

On an organizational level, your company needs to determine the number of people required to maintain its databases and applications. Depending on the size of the databases and the applications, you may have one or more DBAs. They, in turn, may specialize in maintaining databases *or* applications, or they may support *both* a database and the applications that access it.

If you work in a large organization, you are likely to have one user with SYSADM authority who designates several users with DBA authority level and privileges. The DBAs would then share the responsibility for the database maintenance tasks.

If you work in a small organization, the user with SYSADM authority may be the DBA as well.

Read *Chapter 7, Security and Authorization* for more information on authority levels.

Relational databases

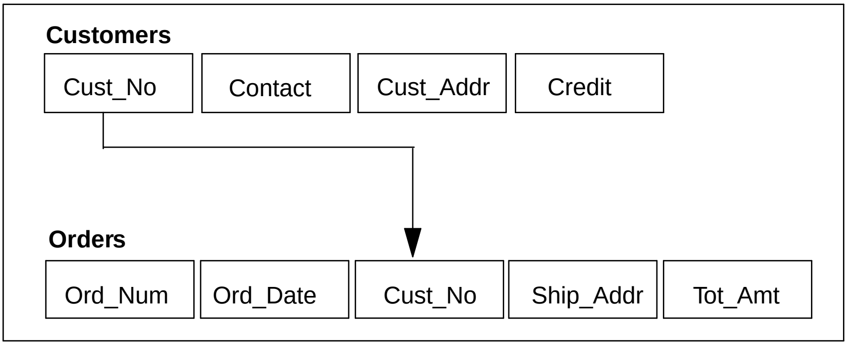
Users need data that is easy to access and that they can use in both ad- hoc queries and daily transaction processing. The data structures should be flexible.

A relational database reduces data to its most basic level in simple, two-dimensional tables that contain columns and rows of data values. Tables are easy to visualize. Users can explain their needs in easy-to-understand terms. Data modeling is flexible.

Representing relationships

A collection of tables can represent complex data relationships. The relationship between two tables is defined by a column that both tables contain. This makes the links between the tables easy to see and understand.

For example, in an order entry system, the customer information is stored in a separate table than the order information. The two tables are related by the existence of a common column: customer number (Cust_No).



Customers and Orders tables in an order entry system

This link enables SQLBase to match a row in the Orders table to its associated row in the Customers table, allowing you to find the customer name and address for each order.

Terminology

The terms used to refer to database objects vary from system to system. The following table shows how the terms relate to each other. Centura uses the "Relational database" terms.

Relational database	**Set theory**	**Conventional data processing**
Table	Relation	File or data set
Row	Tuple	Record
Column	Attribute Domain	Field

The word *relation* comes from the study of data structures done by E.F. Codd and others at IBM. Codd used set theory. A set is a group of objects with a defining rule that enables you to tell whether a given object is in the set or not. For example, the set of white rabbits is a subset of all rabbits with the defining property that the animal has white fur.

A relational system produces a result set by retrieving a set of rows from one or more tables. You can, in turn, operate on this result set instead of operating against the tables from which you originally retrieved the rows. This set-at-a-time approach means that a single relational command can retrieve, update, or delete multiple rows stored in the database.

The sequence (left-to-right or up-to-down) of data in a table holds no information about the data itself. Part of the flexibility of a relational database comes from this property; no information is "hidden" in the physical layout of the database.

In a relational system, the meaning of a column or row does not depend on its order relative to the other columns and rows. This means that you can retrieve data from a select number of columns without having to retrieve data from all columns. The same goes for rows: you can retrieve data from a subset of rows without having to retrieve all the rows.

Relational databases reduce the storage of redundant data. This has three benefits:

- You can rearrange data and combine it easily into new relationships; it is not locked into certain relationships because of the way it is stored.
- You can more easily update data because there are fewer instances of the data. This reduces the likelihood of errors arising from a failure to update all instances.
- It reduces disk space requirements.

Relational operations

Codd's relational algebra defined three key data retrieval operations.

Selection

Selection is the process of retrieving all rows of a table which meet some specified criterion. Note that this is a very narrow definition of selection, and should not be confused with the SQL SELECT command, which does more than just selection.

Projection

Projection is the process of retrieving one or more columns from a table in a specified order and removing any duplicate rows.

Join

Joining is the process of retrieving columns from different tables where the values of a common column in each table are equal.

The ability to join two or more tables is powerful. The join operation distinguishes relational systems from non-relational systems.

The SQL SELECT command performs selection, projection, and join operations.

Comparison to other models

Hierarchical and network database models are procedural and record-at-a-time. To find a record, you have to navigate or find a path to the record you want and give multiple procedural commands that tell the system to walk down that path, step-by-step. You need a detailed understanding of how the data is stored. Once you have created a database and loaded data into it, it can be difficult to change.

In contrast, a relational system provides automatic navigation. You do not have to know how the database stores data in order to retrieve, change, or destroy that data. This makes it easy to access data.

Hierarchical and network database systems store some data as values and other data as pointers. For example, in a network system, the data that order #123 comes to a total of $58.00 would be stored as a value in a field. The data that order #123 belongs to customer #345 would be stored as a pointer from the order record to the corresponding customer record.

You must use pointers to associate records to one another in a network or hierarchical database. You must decide whether to store data as a pointer or as a value when you define the database. Network and hierarchical systems use relationships that are pointer-based and predefined or static.

In contrast, relational database systems can use any value to link one table to another, and you define relationships between values when you query the data, not when you create the tables. This gives you maximum flexibility to create spontaneous queries.

How SQL organizes data

SQL objects include:

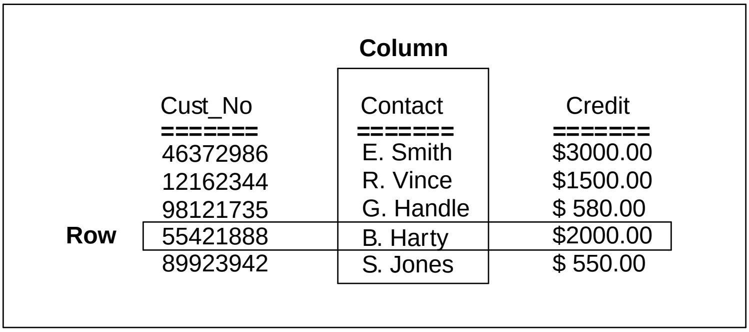
- Databases
- Tables
- Columns
- Indexes
- Views
- Synonyms
- Stored commands
- Stored procedures
- External functions
- Triggers

Databases, tables, and columns

A *database* is a set of SQL objects. When you create a database, you are actually naming an eventual collection of tables, associated indexes, and views.

A single database can contain all the data associated with one application or with a group of related applications. Collecting data into a single database lets you enable or prevent access to all the data in just one operation.

A database contains one or more tables. Each table has a name and contains a specific number of *columns* and *rows*. Each column in a row is related in some way to the other columns in the same row.



Cust_No	Contact	Credit
=======	=======	=======
46372986	E. Smith	$3000.00
12162344	R. Vince	$1500.00
98121735	G. Handle	$ 580.00
55421888	B. Harty	$2000.00
89923942	S. Jones	$ 550.00

Each column has a name and a data type. Data values exist at the intersection of a row and a column.

Data uniqueness

In theory, no row should be a duplicate of any other row in the same table. Consider a sales order table with columns for the date, product code, quantity, and customer ID. If a customer orders ten widgets in the morning and then another ten in the afternoon, this would create two duplicate rows in the table, one for each order.

To ensure the creation of unique rows, you might create an additional column to store the time at which the order was placed, or to store a uniquely-generated sequence number (such as an invoice number).

A table can have a primary key which is a column or a group of columns whose value uniquely identifies each row. Another table can have a foreign key which is a column or a group of columns whose values are equal to those of the first table's primary key. One of the rules of referential integrity is that the value of a table's foreign key must match the value of another table's primary key.

Changing the database structure

SQLBase has SQL commands that enable you to add new columns to existing tables and make existing columns wider. These change take effect immediately and no database reorganization is necessary.

Indexes

An *index* is an ordered set of pointers to the data in a table. It is based on the data values in one or more columns of the table. SQLBase stores indexes separately from tables.

SQLBase decides whether or not to use an index when accessing a table, and users need not be aware that SQLBase is using an index.

An index provides two benefits:

- It improves performance because it makes data access faster.
- It ensures uniqueness. A table with a unique index cannot have two rows with the same values in the column or columns that form the index key.

Views

A *view* is an alternate way of representing data that exists in one or more tables. A view can include all or some of the columns from one or more *base tables*. You can also base a view on other views or on a combination of views and tables.

A view looks like a table and you can use it as though it were a table. You can use a view name in a SQL command as though it were a table name. You cannot do some operations through a view, but you do not need to know that an apparent table is actually a view.

A table has a storage representation, but a view does not. When you store a view, SQLBase stores the definition of the view in the system catalog, but SQLBase does not store any data for the view itself because the data already exists in the base table or tables.

A view lets different users see the same data in different ways. This allows programmers, database administrators, and end users to see the data as it suits their needs. For example, employees might have access to columns containing general information about office locations and phone extensions, while administrators might have access to additional columns containing more confidential information such as employees' home addresses and phone numbers.

Synonyms

A *synonym* is another name for a table, view, or external function. When you access a table, view, or external function created by another user (once you have been granted the privilege), you must fully-qualify the table name by prefixing it with the owner's name, unless a synonym for the table or view is available. If one is available, you can refer to the user’s table or view without having to fully qualify the name.

Stored commands and procedures

A *stored command* is a compiled query, data manipulation command, or procedure that is stored for later execution. SQLBase stores the command’s or procedure’s execution plan as well, so subsequent execution is very fast.

A SQLBase procedure is a set of Scalable Application Language (SAL) and SQL statements that is assigned a name, compiled, and optionally stored in a SQLBase database. Procedures reduce network traffic and simplify your applications since they are stored and processed on the server. They also provide more flexible security, allowing end users access to data which they otherwise have no privilege to access.

SQLBase procedures can be static or dynamic. *Static procedures* must be stored (at which time they are parsed and precompiled) before they are executed. *Dynamic procedures* contain dynamic embedded SQL statements, which are parsed and compiled at execution time.

SQLBase also provides preconstructed procedures as useful tools to help you maintain your database. See *Appendix B* for a description of SQLBase-supplied procedures.

External functions

An *external function* is a user-defined function that resides in an “external” DLL (Dynamic Link Library) that is invoked within a SQLBase stored procedure. SQLBase accepts external functions in the language of your choice, such as C and C++. The SQLBase server converts data types of parameters that are declared in stored procedures into their external representation.

Using external functions enhances the power of the SQLBase server, allowing you to achieve maximum flexibility and performance with minimal programming effort. It extends the functionality of stored procedures with no impact on the application or the server. When external functions are called, they are dynamically plugged in and behave like built-in functions. For details, read *Chapter 8*, *External Functions* in the *SQL Language Reference*.

Triggers

A *trigger* activates a stored or inline procedure that SQLBase automatically executes when a user attempts to change the data in a table. You create one or more triggers on a table, with each trigger defined to activate on a specific command (an INSERT, UPDATE, or DELETE). You can also define triggers on stored procedures.

Triggers allow actions to occur based on the value of a row before or after modification. Triggers can prevent users from making incorrect or inconsistent data changes that can jeopardize database integrity. They can also be used to implement referential integrity constraints. For details on referential integrity, read *Chapter 6, Referential Integrity* in the *SQL Language Reference*.

For details on the trigger execution order before a single data manipulation statement is executed, read the Section *DML Execution Model* in *Chapter 1* of the *SQL Language Reference*.

System catalog tables

SQLBase maintains a system catalog, or *data dictionary*, in every database. The system catalog contains tables that store information about the database's tables, views, columns, indexes, execution plans, and security privileges.

When you create, change, or drop a database object, SQLBase modifies rows in the system catalog tables that describe the object and how it is related to other objects.

The system catalog also contains the name, size, type, and valid values of each table's columns.

You can query the system catalog tables just like any other table.

Read *Appendix A, System Catalog Tables* for detailed information about the data dictionary tables.

Concurrency and consistency

To guarantee the integrity of data, SQLBase uses page locks to prevent users from changing data while another user is reading or changing data on the same page. Locks prevent lost updates from occurring and ensure that data does not change while you are reading it or changing it yourself.

SQLBase stores data in 1K pages, and can lock both base table pages and index pages. For details, read the section *Database locking* in *Chapter 4, About Databases*.

Databases and networks

This section describes client/server terminology as it applies to SQLBase.

Distributed processing

The term *distributed processing* describes the placement of computers where they are needed, whether on the floor of a manufacturing facility, in an accounting department, in a laboratory, at a field location, or in an executive office. A network links computers so that they can exchange information, share resources, and perform distributed processing.

A distributed processing network allows information to originate anywhere in the network. You can place systems where they are needed while still having access to the facilities of other widely dispersed systems. Information can be exchanged among all parts of an organization.

The term distributed processing had been used for several years to label network configurations where remote computers exchanged data. Two developments in the 1980s brought a new meaning to the term: PCs (Personal Computers) and LANs (Local Area Networks).

PCs supplied solutions to many simple information processing requirements. The user interface on PCs became sophisticated with bit-mapped displays that used windowing and graphics, and a mouse. This made applications easy to use.

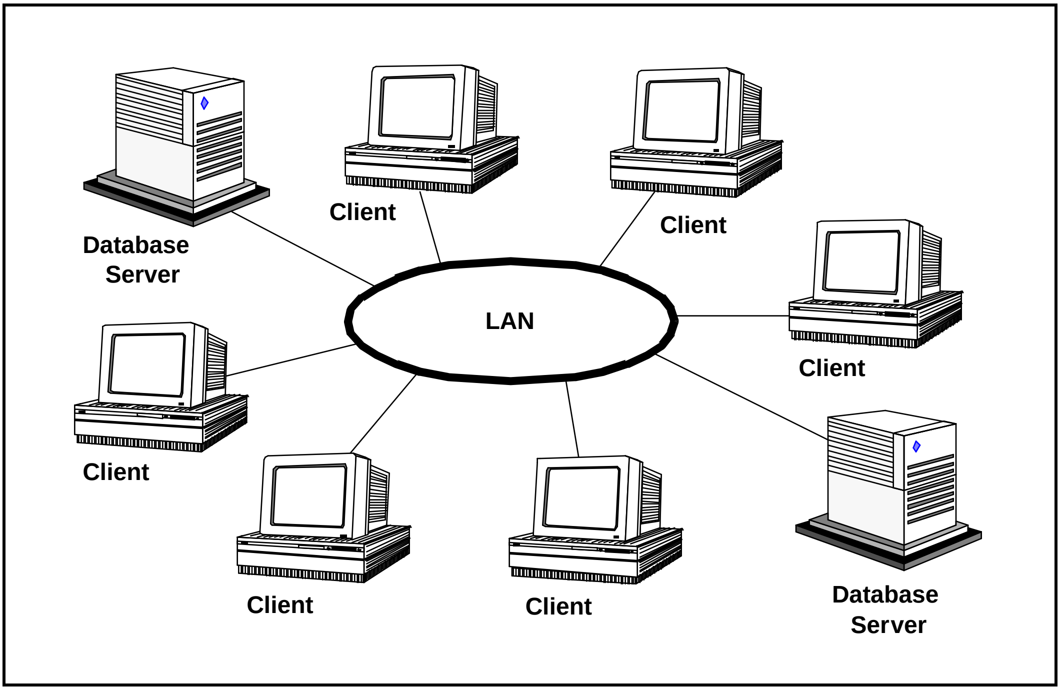
With a LAN, PCs could communicate with each other and share resources such as databases. Users began to wonder if they could move some transaction processing systems, which had traditionally run on minicomputers and mainframes, onto PCs. They also began to think about new types of applications that had not been possible before.

The client-server model

Client-server technology is a form of distributed processing where computing activities are shared among cooperating, networked computers. An application is functionally split into two or more programs which execute on different computers and communicate with each other by passing messages across the network.

The client programs run on users' PCs. Server programs run on more powerful computers. The client sends a request to a server and when the server receives the request, it processes it and sends the results back to the client.

Typically, a local-area network contains more clients than servers. Client computers cannot share their resources or use the resources of other client computers.



Through client-server architecture, users gain access to capabilities available on a server computer which are not available locally on their desktop computer. A client-server application has the capabilities of both the client and the server.

Servers are usually the most powerful computers on the local area network. Certain criteria make some systems better suited to be servers, such as a high input/output transfer rate and network throughput (the speed at which a server processes network tasks). High disk capacity and multiple disk controllers are another criteria. A server is usually multi-tasking so that it can serve multiple clients simultaneously.

Cooperative processing

When the processing duties are shared by a client and server in a sophisticated manner, it is called *cooperative processing*. An example of cooperative processing is a distributed database system. A client takes user input and submits processing requests to the server. Database processing, such as retrieving and sorting data, is centralized at a database server. Application processing is distributed between the clients and the server.

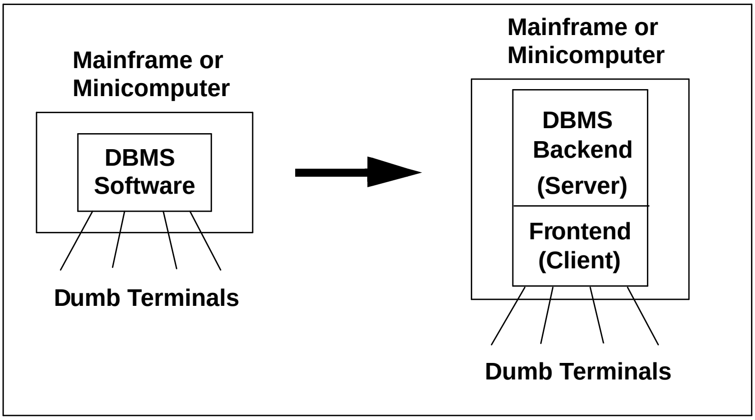
The client and server cooperate to do a job. The client manages the screen display and graphics, carries on an interactive dialogue with the user, validates data, does some local computation, and provides a network interface that talks to the server. The database server is responsible for data storage, security and integrity, and input/output.

In a cooperative processing environment, components of an application run where they are best suited. Data-intensive applications suffer performance problems when all computing happens on the client. The idea is to off-load time- and data-intensive tasks to the database server while doing as much processing on the client as possible. It is often the case that requests are processed faster and more economically on a shared high-performance database server than on a standalone client computer.

By their nature, some applications are suited to cooperative processing. For example, a *network-intrinsic* application is one that does not make sense on a standalone system. Communications is an integral part of the overall process of completing the application. Distributed database is a good example of a network-intrinsic application because it uses the network to provide *shared* access to data for many users.

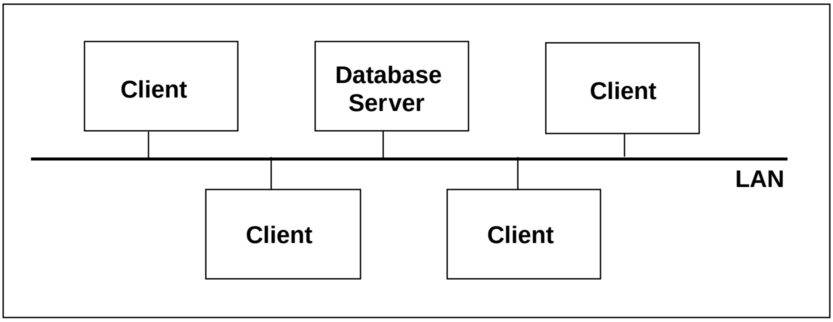
Distributed database

Database software has evolved. At first, database software was monolithic and ran on one large computer accessed by dumb terminals. Later, database software become more sophisticated and split the database function into two components: the client (frontend) and the server (backend). Note that both components still ran on the same mainframe or minicomputer at this time.



The start of database software evolution

Eventually, database software vendors moved the client component down to microcomputers and LANs, and the database server soon followed.



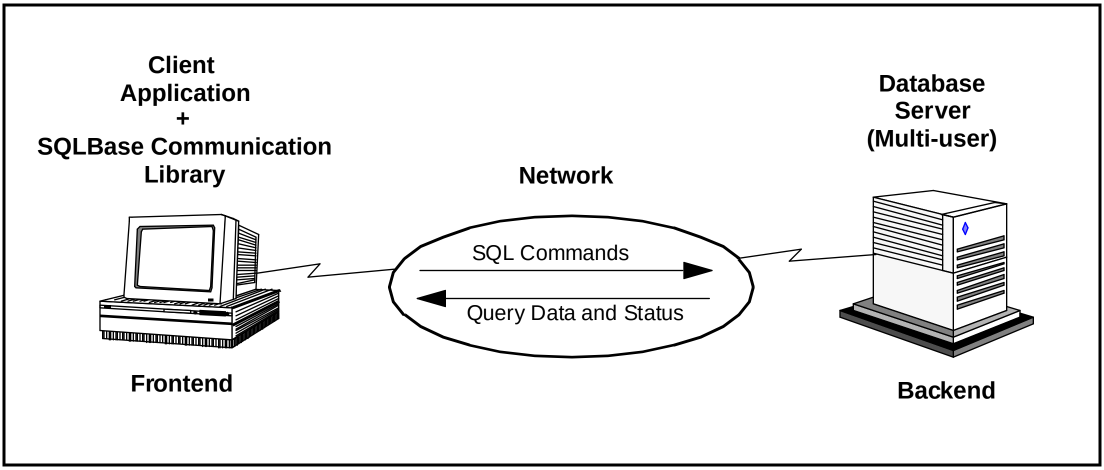
A database server and clients on a LAN

The difference from file servers

Database servers are different from file servers. File servers are responsible for storing, not processing, data files. Application processing is the client's responsibility and a client's request for data results in the file server sending an entire database across the network to the client. As the number of clients making requests for data increases, so does the likelihood of network performance degradation and bottlenecks.

In contrast, a relational database system has the flexibility and functionality necessary to work across networked computer systems. A client application uses SQL to talk to a database server. The client is responsible for executing the application and formatting a compact SQL command which it then sends across the network to the database server. The database server is responsible for parsing and executing the SQL command and sending data and a status code back to the client.

Unlike a file server, a database server sends only a part (subset) of the database back to the requesting PC. Most database servers use SQL because it is a convenient language for specifying logical subsets of data.



Cooperative processing: a client and server communicating via SQL

Benefits of distributed databases

The benefits of distributed databases are:

- Location independence. The location of a client application is independent of the location of the data.
- Location transparency. Users can access a database without having to know its location, and you can move a database with no affect on users or applications.
- Incremental application growth. You can upgrade an application by adding more clients or by purchasing a more powerful database server machine.
- Site autonomy. You can maintain each database separately from other databases and you can administer the shared data and allow programmers to support the applications.
- You can distribute and secure data to match individual departmental needs.
- Hardware and software independence. You can use hardware and software from different manufacturers, and you can match them to the requirements of your applications.
- Distributed processing. You can distribute the processing and storage of data among many computers.
- Increased availability and reliability. If a database server goes down, only its clients are affected, not your entire organization.

Chapter 2

Communications

Although it is not usually a DBA’s responsibility to configure network software, you do need to know how SQLBase uses network resources and how to configure SQLBase for various communications options.

Communication requirements

SQLBase software consists of both client and server components.

Client side software consists of:

- Applications (like SQLTalk)
- SQL/API
- Communication libraries to remote servers

Server side software consists of:

- Single and multi-user database servers
- Communication libraries

Single-user configuration

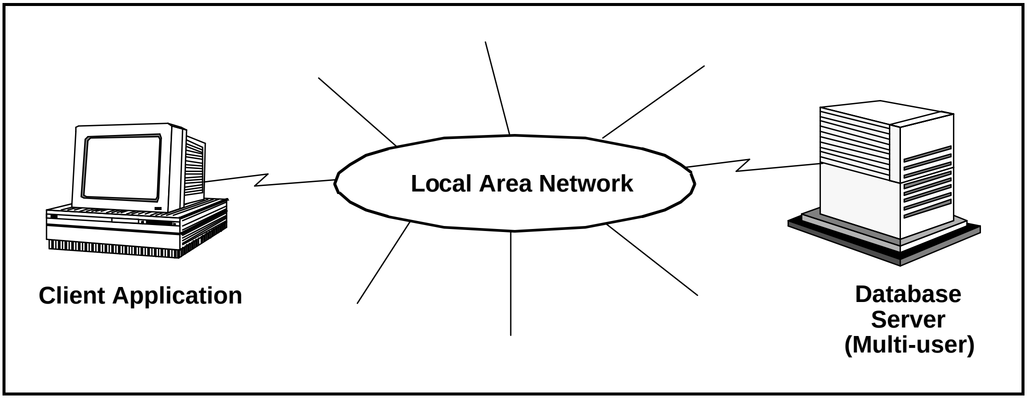
In a single-user configuration, you need only one computer. You run an application and a single-user database server on the same machine. The database server accesses local databases only.

Multi-user configuration

In a multi-user configuration, you need at least two computers because a server and its clients are separate nodes on a local area network (LAN). The client and the server communicate with each other through a communications interface such as NetBIOS.

You run an application and communication library on the client side, and a multi-user database server and communication library on the server side.

The communication libraries on both sides enable Centura client applications and SQLBase servers to communicate with each other over a LAN.



A multi-user configuration: a client and server communicating across a LAN

The multi-user configuration is dependent on your having installed local area network software on both the client and server computers. You should verify that the network software is loaded and working before attempting to connect a client and server.

SQLBase supports the following network protocols:

- NetBIOS
- Anonymous Pipes
- SPX
- TCP/IP

See the *Client-to-server communications matrix* on page *2-18* to learn which clients can connect to which servers and to discover the protocol options that support each configuration.

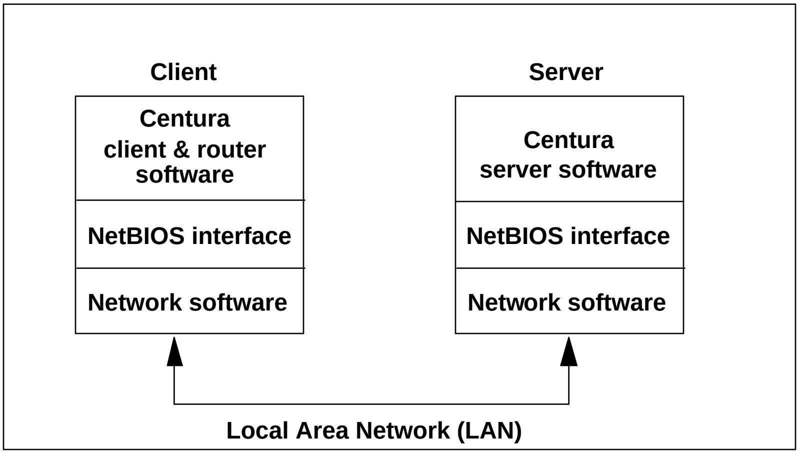
NetBIOS

NetBIOS (Network Adapter Basic Input/Output System) is a standard networking interface that is made available by many network software packages. It provides the interface between networking services and a computer's operating system.

SQLBase can run with any network software package that supports a standard NetBIOS interface. SQLBase interfaces with the network software at the NetBIOS API (Application Programming Interface) level and NetBIOS handles all the lower-level details (such as network addresses).

SQLBase supports NetBIOS for Windows 3.x, Windows NT, and Windows 95.

This diagram shows how SQLBase uses NetBIOS to interface to the network software.



NetBIOS commands

SQLBase uses the following NetBIOS commands:

Type	Command Code	Description
General		
	0x32	Reset local adapter
	0x33	Get adapter status
	0x35	Cancel command
Name Support		
	0x30	Add unique name
	0x31	Delete name
	0x36	Add group name
Session Support		
	0x10	Call; open a session
	0x11	Listen; enable a session
	0x12	Hang up; close a session
	0x14	Send data
	0x15	Receive data

NetBIOS implementation

Because the information necessary to implement a NetBIOS interface is available to all network vendors, many vendors do provide a NetBIOS interface for their networks. Their implementations differ only in the protocols they choose to implement the transport and network layers.

Applications using NetBIOS don’t actually send and receive messages; they make requests to send and receive messages. NetBIOS then relies on the vendor’s network protocol of choice to provide communications support. IBM NetBIOS, for example, uses the NetBEUI and XNS protocols while Novell NetBIOS uses the IPX protocol.

For this reason, you should have the same network software installed on the client and server computers. Both sides must be able to receive and interpret the chosen network protocol.

NetBIOS communication libraries

These are the Centura NetBIOS communication libraries:

Platform	Communication Libraries
Windows	*sqlnbiow.dll*
Windows NT	*sqlntnbi.dll*
Windows 95	*sqlntnbi.dll*

On these platforms, NetBIOS support is implemented via dynamic link libraries (DLLs).

IBM NetBIOS vs. non-IBM NetBIOS

Centura supports two versions of NetBIOS:

- IBM NetBIOS

  This is provided by IBM with network products such as IBM LAN Server, IBM LAN Requester, and IBM Communications Manager.

- Non-IBM NetBIOS

  This is provided by non-IBM network vendors with network products such as Novell NetWare and Microsoft LAN Manager.

Centura's IBM NetBIOS support uses the 'NETBIOS' function found in the *acsnetb.dll* file provided with IBM network software. Centura's non-IBM NetBIOS support uses the 'NetBiosSubmit' function as well as related functions found in the *netapi.dll* file provided with non-IBM network software.

Windows 3.x platform. Centura provides a single NetBIOS communications library (*sqlnbiow.dll*) that contains the necessary functions for both IBM and non-IBM NetBIOS.

Windows NT platform. Centura provides a single NetBIOS communications library (*sqlntnbi.dll*) that contains the necessary functions for both IBM and non-IBM NetBIOS.

Windows 95 platform. Centura provides a single NetBIOS communications library (*sqlntnbi.dll*) that contains the necessary functions for both IBM and non-IBM NetBIOS.

Be sure to use the type of NetBIOS support appropriate for your network software.

NetBIOS resources

Three NetBIOS resources are important to SQLBase: *sessions*, *commands*, and *names*.

A NetBIOS *session* is a connection between two applications which exchange a series of messages in order to complete a task. A session can exist for extended periods. Such a connection is sometimes called a virtual circuit. A multi-user database server can have multiple simultaneous sessions servicing many clients.

During a network session, NetBIOS *commands* are issued. Some NetBIOS commands execute in *non-blocking* mode (also called no-wait mode). When a NetBIOS command executes in non-blocking mode, the program does not wait until the command completes. Before the command completes, another NetBIOS command can be issued. This is why a single NetBIOS session can simultaneously use more than one command.

A NetBIOS name is allocated to the server and to each database that resides on the server. The names of the server and the databases are put into a NetBIOS name table on the server machine. These names are then broadcast on the network and serve to identify the machine to client applications that specify a database name in order to establish a session.

How SQLBase uses NetBIOS resources

SQLBase clients and database servers use NetBIOS resources differently.

Client

On the client's side, each application that connects to a database uses one session (regardless of the number of cursors that the application connects) and at least one, but no more than two, commands.

For example, a process uses an extra command when an application switches between cursors. Therefore, *commands* should always have a higher value than *sessions*. In general, the following is a good rule of thumb:

```
commands = sessions + 20%
```

Server

On the server's side, NetBIOS resource allocation is more complex.

- The server program itself uses one session, one command, and one name.
- Each database on the server uses one session, one command, and one name.

  If you have many databases, you may need to increase the number of names available to SQLBase as described in the next section. Note that you can conserve NetBIOS resources by having fewer databases. If possible, combine the tables from two or more databases into one.

- Each client connection uses one session (regardless of the number of cursors that the application connects) and at least one, but no more than two, commands.

  For example, a process uses an extra command when an application switches between cursors. Therefore, *commands* should always have a higher value than *sessions*. In general, the following is a good rule of thumb:

  ```
  commands = sessions + 20%
  ```

- Each new cursor connection temporarily requires a session. If the cursor belongs to an application that already has a session established, the temporary session is freed and the pre-existing session is shared by the new cursor.

If a multi-user server program provides access to more than one database, be sure to allocate enough NetBIOS resources for the network software.

A server’s Process Activity window displays an error message if too few resources are allocated. Other symptoms include the database name disappearing from the Databases window or the user on the client computer receiving a “cannot open database”, “network not functioning”, or “name table full” error

How to set NetBIOS resources

This section shows how to set the number of NetBIOS sessions, commands, and names for widely-used network software packages.

3Com 3+Open

3Com 3Plus NetBIOS looks in *ldr.cfg* for the number of sessions and commands. To define 32 sessions and 64 commands, specify:

```
program=nba.exe ... /s32 /n64
```

3Com LAN Manager

3Com LAN Manager looks for a configuration file named *protocol.ini*. To define 32 sessions and 64 commands in *protocol.ini*, specify:

```
MAXSESS=32
MAXNCB=64
```

IBM LAN Server and Requester

Setting network resources for IBM LAN Server or Requester is a three-step process.

First, allocate the system-wide NetBIOS resources by editing the *protocol.ini* file. This creates a pool of NetBIOS resources that can be used by all the processes on the machine.

Second, allocate IBM NetBIOS resources to IBM's LAN Server or Requester software with the *net1* keyword in the [networks] section of the *ibmlan.ini* file:

```
net1=driver$, adapter, version, sessions, commands, names
```

where:

*driver$* is the network device driver. This value is set during LAN Server or LAN Requester installation. Examples of values include 'netbios$' and 'netbeui$'.

adapter is the number of the network adapter (0 or 1).

version is the driver version number. Examples of values include 'LM10' and 'NB30'.

sessions is the number of NetBIOS sessions that LAN Server or Requester can maintain.

commands is the number of simultaneous NetBIOS commands LAN Server or Requester can post.

names is the number of NetBIOS names that LAN Server or Requester can register.

For example:

```
net1=netbios$, 0, NB30, 32, 64, 16
```

Third, allocate NetBIOS resources to SQLBase by setting the values of the *sessions*, *commands*, and *names* keywords in the appropriate section of the configuration file (sql.ini). For instructions, read *Chapter 3, Configuration (sql.ini)*. Remember that the combined total of resources requested for LAN Server or Requester (in *ibmlan.ini*) and for SQLBase (in sql.ini) must be less than or equal to the system-wide total set in *protocol.ini*.

IBM Token Ring NetBIOS

IBM Token Ring NetBIOS looks in *config.sys* for the number of sessions and commands. Increase the number of NetBIOS sessions and commands on the IBM Token Ring NetBIOS 'DEVICE' driver line in the *config.sys* file.

Microsoft LAN Manager

Allocate NetBIOS resources for the NetBIOS device driver in the [netbeui] section of the *protocol.ini* file. For example:

```
[netbeui]
commands=64
sessions=32
```

When initializing NetBIOS, SQLBase uses the LAN Manager default which is *net1*. Allocate a global pool of NetBIOS resources using the *net1* keyword in the [networks] section of the *lanman.ini* file:

```
net1=driver$, adapter, version, max_sessions, max_commands
```

where:

driver$ is the network device driver. This value is set for you by the LAN Manager installation program.

adapter is the number of the network adapter. This value is set for you by the LAN Manager installation program.

version is the LAN Manager version number (in capital letters). Examples are 'LM10' or 'NB30'.

max_sessions is the maximum number of simultaneous NetBIOS sessions.

max_commands is the maximum number of simultaneous NetBIOS commands.

For example:

```
net1=mnetb$, 0, LM10, 40, 80
```

Note: Starting the redirector (Net start rdr) reserves four NetBIOS commands. Applications communicating via NetBIOS cannot use additional command blocks reserved for the redirector by the *lanman.ini*'s MAXCMDS parameter. The default of MAXCMDS is 16.

In the example, the maximum number of NetBIOS commands available to application programs is: 80 - 4 - 16 = 60.

Allocate NetBIOS resources for SQLBase in the sql.ini file.

LAN Manager is NDIS-compatible.

Note: Windows clients accessing a SQLBase database via SPX must still load NETX.

NetBIOS configuration options

These configuration keywords are specific to Centura products using NetBIOS:

adapter

commands

ibmnetbios

listenretry

names

reducethread

retry

retrytimeout

sessions

serverprefix

Read *Chapter 3, Configuration (sql.ini)* for complete descriptions of these keywords.

NetBIOS errors

SQLBase passes on NetBIOS errors and displays their error numbers in the 7000 and 9000 ranges.

If you get an error in one of these ranges, use the following procedure to find the actual NetBIOS error value:

1. If in the 7000 range, subtract 7000.
   If in the 9000 range, subtract 9000.
2. Convert the value to hexadecimal.

Assume, for example, that SQLBase returns error 7018:

1. 7018 - 7000 = 18
2. 18 decimal = 12 hexadecimal

Now look up the hexidecimal value of the NetBIOS error in any NetBIOS manual:

NetBIOS error	Description
12	Session Open Rejected because No Listen is Outstanding.

Anonymous Pipes

Anonymous Pipes is a communication protocol that is local only to Windows NT and Windows 95. SQLBase supports Anonymous Pipes for both platforms.

Anonymous Pipes communication libraries

The Centura Anonymous Pipes communication library for Windows NT and Windows 95 is *sqlapipe.dll*.

Anonymous Pipes configuration options

Anonymous Pipes is local-only for Windows NT and Windows 95. There are no specific keywords for Anonymous Pipes on these platforms.

SPX

SPX (Sequenced Packet eXchange) is a Novell communication protocol derived from Novell's Internetwork Packet eXchange (IPX). SQLBase writes to the SPX interface protocol which in turn uses IPX to send data packets.

Variables such as packet size are defined by Novell's SPX and IPX interface protocols, and SQLBase has no control over these settings.

SQLBase supports SPX for the NetWare, Windows 3.x, Windows NT, and Windows 95 server platforms.

SPX communication libraries

These are the Centura SPX communication libraries:

Platform	Communication Libraries
NetWare 3.x	*spxdll.nlm*
NetWare 4.x	*spxdll40.nlm*
Windows 3.x	*sqlspxw4.dll*
Windows NT	*sqlwsspx.dll*
Windows 95	*sqlspx32.dll*

SPX configuration options

These configuration keywords are specific to Centura products using SPX:

commonname
insertioncontext
ndsloginid
ndsloginpassword
nwadvertisemode
numdb
numsessions
preferrednameservice
retry
retrytimeout

Read *Chapter 3,Configuration (sql.ini)* for complete descriptions of these keywords.

TCP/IP

SQLBase supports TCP/IP connections on the NetWare platform through Novell's Transport Layer Interface (TLI) API. With this feature, you can connect your client applications using TCP/IP to SQLBase servers using NetWare. SQLBase also supports TCP/IP connections between NetWare API clients and SQLBase servers.

TLI supports data transfer between two transport endpoints. One endpoint is a transport user and the other is a transport provider. The transport provider can use different protocols; For example, Netware uses SPX/IPX or TCP/IP. Currently, SQLBase only supports TCP/IP with TLI.

The NetWare server can listen simultaneously on both SPX and TCP/IP protocols. You can have two clients connecting to the same server; one using SPX and one using TCP/IP.

To enable both protocols simultaneously on the server, specify both comdlls in the [*servername*.dll] section, where *servername* is the name of the server in use. You will also need to load both of the protocol Netware Loadable Modules (NLMs) on the NetWare server.

Prerequisites

To use this new feature, you must install the TCP/IP protocol on the NetWare machine. You can obtain the software directly from Novell.

TCP/IP communication libraries

These are the Centura TCP/IP communication libraries:

Platform	Communication Libraries
NetWare	*tlidll.nlm*
Windows 3.x	*sqlwsock.dll*
Windows NT	*sqlws32.dll*
Windows 95	*sqlws32.dll*

TCP/IP configuration options

These configuration keywords are specific to Centura products using TCP/IP:

listenport	*preferrednameservice*
searchcontext	*serverpath*

For complete descriptions of these keywords, read *Chapter 3, Configuration (sql.ini).*

SNMP

This section describes how SQLBase uses the SNMP protocol. This protocol is only valid if you are using SQLConsole. SQLConsole uses SNMP to retrieve information from the SQLBase server.

What is SNMP?

The Simple Network Management Protocol (SNMP) is an open standard protocol that specifies how information is exchanged between devices on a network. The Internet Engineering Task Force (IETF) has adopted SNMP as the standard network protocol for managing network devices. The simplicity of SNMP implementations and its endorsement by the IETF have contributed to the widespread use of SNMP in network management tools.

SNMP is implemented as an asynchronous protocol that is not dependent upon the network transport mechanism. Current implementations of SNMP can operate over TCP/IP and Novell IPX/SPX.

What does SNMP monitor?

SNMP is commonly used to retrieve information and manage a variety of network devices (such as hubs and bridges) from different vendors and on different platforms from a central location. For example, consider the following configuration:

- A bridge to a Novell Ethernet network with several Novell Netware file servers and a Windows/NT SQLBase database server also offering file services.
- Several hundred workstations.

Using SNMP, you can retrieve information both from the various network devices and also the servers on the network. If the devices networks have the same MIB (described on the following page), the information returned will be consistent and comparable regardless of vendor or platform. The management tool need only understand SNMP and definitions about what information can be managed. This eliminates the need for several management tools in a multi-vendor environment.

In addition to monitoring hardware, the flexibility of SNMP allows you to monitor and manage virtually anything that communicates across a network, including software program. For example, SQLConsole uses SNMP to retrieve information from the SQLBase server.

Components

There are three key components in an SNMP managed network:

- The MIB

  The Management Information Base is the definition of what information is available, the description of that information, how it is accessed, and how it should be displayed. Objects in the MIB are identified by their Object Identifier (OID).

- The Agent

  The agent provides information to satisfy SNMP requests for information pertaining to the managed device. The agent may also issue notification messages or traps to another device on the network, typically a management station. An SNMP agent is required on each managed device.

- The NMS

  The Network Management Station is a network-connected workstation that can run software tools such as Novell NMS, Hewlett-Packard's OpenView or an SNMP enabled custom application.

  The NMS uses SNMP to issues requests, or *polls*, for information to managed devices on the network. Sometimes, the SNMP agent can volunteer this information on its own, without having to receive a request first.

The NMS uses SNMP to *poll* information from specific network device addresses. These SNMP requests contain both an OID (Object Identifier) that specifies which information is to be retrieved, and a network address specifying a device to receive the query. The NMS consults the MIB to establish what it can ask about and to determine how to package the request. Upon receiving an SNMP request, the agent on the target device consults its MIB and responds by returning the requested information if the agent is capable of satisfying the request.

How SQLBase and SQLConsole use SNMP

The SQLBase SNMP agent can provide information about a SQLBase server connected to a network. The information that can be monitored is described in the RDBMS MIB. This MIB is located on your SQLConsole installation diskettes and is installed during SQLConsole installation.

The SQLBase agent

The SQLBase agent is a logical entity (process, thread, or extension to another program) running on a system that also runs a SQLBase database server. The agent is responsible for satisfying queries according to the definition of the RDBMS MIB.

The Network Management Station

The SQLBase implementation of SNMP provides an add-in program that extends the functionality of HP OpenView for Windows. This program provides the RDMBS MIB information specifically for the SQLBase Windows/NT SNMP Agent.

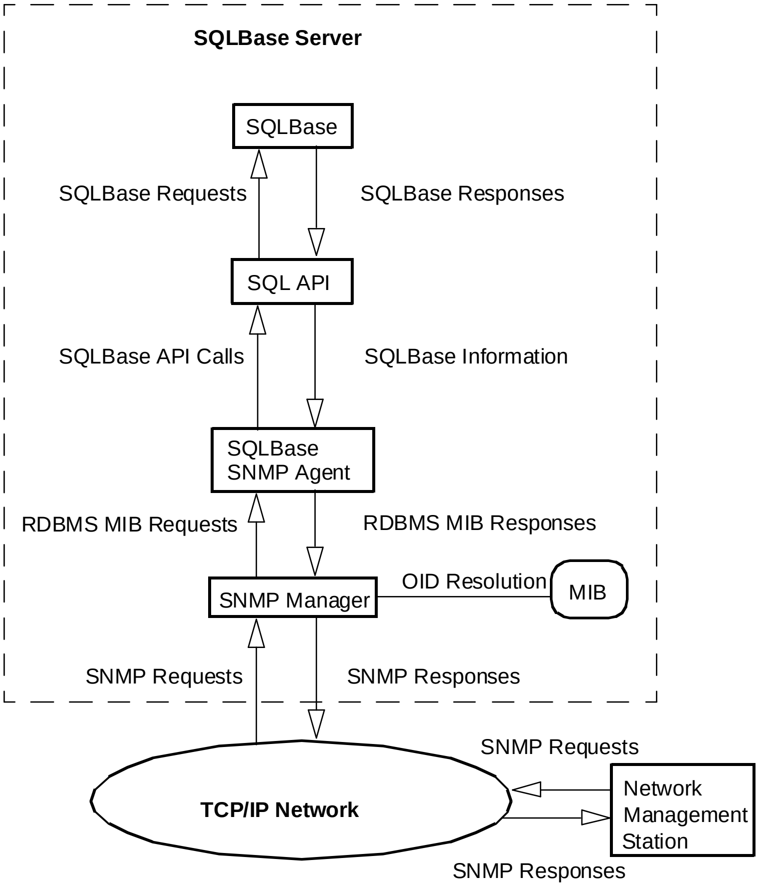
Any NMS communicating over a network using TCP/IP can query the SQLBase for Windows/NT SNMP Agent software.

The HP OpenView program provides the following features:

- Automatically recognizes the SQLBase SNMP Agent during discovery.
- Presents RDBMS information in a SQLBase specific manner.
- Launches SQLConsole from HP OpenView.

SNMP flow diagram

The following diagram illustrates the general architecture and communication flow of the SQLBase SNMP Agent.



The SQLBase MIB allows you to monitor several servers at one network address. However, since only one instance of SQLBase can be executing on a given server at a time, information will only be available for one SQLBase database server. The SQLBase Database SNMP Agent only provides information about SQLBase; it does not provide information about databases from any other vendors.

Currently only the SQLBase Database Server for Windows/NT is supported by an SNMP agent. The Windows/NT SNMP services use TCP/IP to communicate across the network. The server and the NMS must therefore have TCP/IP installed and operational. There is no requirement for other SQLBase client workstations to use TCP/IP unless they are SNMP capable and wish to access the SQLBase SNMP agent.

Client and server communication options

You can connect a client application to a local or remote SQLBase database. Local means that the client application and database reside on the same computer, while remote means that they reside on different computers.

On most platforms, you can access a SQLBase database with either a single-user database server or a multi-user database server.

The following sections provide details about the various communications options available on each client and server platform:

- The *Single-user database servers* sections describe client options for connecting to local databases.
- The *Client communication libraries* sections describe client options for connecting to remote databases.
- The *Multi-user database servers* sections describe server options that enable clients to connect to remote databases.
- The *Server communication libraries* sections describe server options for communicating with clients.

Be aware that what we mean by *communication libraries* are those files that provide network protocol support, as well as dynamic link libraries (DLLs).

Protocols supported by platform

This table summarizes the platforms on which SQLBase runs and the protocols supported by SQLBase on each platform. Note that in some cases, either the client or server software supports a protocol that the other does not.

Platform	Client or Server	Protocols Supported
NetWare	Client	SPX, TCP/IP
	Server	SPX, TCP/IP
Windows	Client	Interprocess, NetBIOS, SPX, TCP/IP
	Server	Interprocess
Windows NT	Client	NetBIOS, Anonymous Pipes, SPX, TCP/IP

Platform	Client or Server	Protocols Supported
	Server	NetBIOS, Anonymous Pipes, TCP/IP
Windows 95	Client	NetBIOS, Anonymous Pipes, SPX, TCP/IP
	Server	NetBIOS, Anonymous Pipes, TCP/IP

Client-to-server communications matrix

The following table shows valid client and server communications, and their supported protocol options.

For example, a Windows NT client can communicate with a:

- NetWare server via either SPX or TCP/IP.
- Windows NT server via NetBIOS, Anonymous Pipes, or TCP/IP.
- Windows 95 server via NetBIOS or TCP/IP.

CLIENTS

	PLATFORM	***NetWare***	***Windows***	***Windows NT***	***Windows 95***
SERVERS	***NetWare***	SPX TCP/IP	SPX TCP/IP	SPX TCP/IP	SPX TCP/IP
	Windows		Inter-process		
	Windows NT	TCP/IP	NetBIOS TCP/IP	NetBIOS Anonymous Pipes TCP/IP	NetBIOS TCP/IP
	Windows 95	TCP/IP	NetBIOS TCP/IP	NetBIOS TCP/IP	NetBIOS Anonymous Pipes TCP/IP

Matrix of SQLBase client/server communication protocols by platform.

Microsoft Windows communication options

A Microsoft Windows client can connect to multiple database servers simultaneously via *sqlapiw.dll*.

Sqlapiw.dll is a general communications interface library that connects Windows applications to local and remote SQLBase servers. *Sqlapiw.dll* makes calls to communication libraries to perform functions.

Communication libraries specify how an application and a database server will communicate. Communication can take place between an application and a database server on the same computer (interprocess communication) or between an application and a database server across the network. In the case of remote communication, Centura provides protocol-specific communication libraries.

You tell *sqlapiw.dll* which communication libraries to load with one or more *comdll* keywords in the [winclient.dll] section of the client's configuration file (sql.ini):

```
[winclient.dll]
comdll=communications_library
comdll=communications_library
comdll=communications_library
```

Sqlapiw.dll reads this section of the configuration file when it is initialized by a client application.

If you specify two or more *comdll* keywords in the [winclient.dll] section, an attempt to connect to a database causes *sqlapiw.dll* to use each communication library in turn until it finds the database.

For example, consider the following configuration file:

```
[winclient.dll]
comdll=sqldbw
comdll=sqlnbiow
```

The application first looks for the database locally because *sqldbw.dll* is an interprocess communications library. If it is unsuccessful, it then looks remotely, searching the network via NetBIOS because *sqlnbiow.dll* is a NetBIOS-specific communications library.

Single-user database server

In windows, your application can access a local database using a single-user database server.

dbwservr.exe

This is a single-user multiple-application server for Windows. You can connect more than one local application to a database via *dbwservr.exe*.

To load *dbwservr.exe*, specify:

```
comdll=sqlwsv
```

in the [winclient.dll] section of a client's configuration file (sql.ini). Select **File**, **Run...** from the Program Manager. Then, type:

```
c:\centura\dbwservr
```

in the dialog box.

You can also load dbwservr.exe into memory by clicking on its icon from the Program Manager.

Server communication library

On Windows, the database server accesses only local databases, so there are no separate protocol-specific server communication libraries necessary.

Client communication libraries

SQLBase Windows client software supports NetBIOS, SPX, and TCP/IP.

- With NetBIOS, clients can access SQLBase for Windows NT and Windows 95 database servers.
- With SPX, clients can access SQLBase for NetWare database servers.
- With TCP/IP, clients can also access SQLBase for NetWare, Windows NT, and Windows 95 database servers.

sqlnbiow.dll

This is a NetBIOS communication library. When you specify:

```
comdll=sqlnbiow
```

in the [winclient.dll] section of a client's configuration file (sql.ini) and try to access a remote database via a Windows application, *sqlnbiow.dll* is loaded into memory.

These keywords are specific to Windows NetBIOS communications and can be configured in the [winclient.nbiow] section of the client's configuration file (sql.ini) :

adapter *retry* *serverprefix*
ibmnetbios *retrytimeout*

Note: The *ibmnetbios* keyword is relevant only for IBM LAN Server environments.

Read *Chapter 3, Configuration (sql.ini)* for complete descriptions of these keywords.

sqlspxw4.dll

This is an SPX communication library used in conjunction with the NetWare DOS Requester (VLM). When you specify:

```
comdll=sqlspxw4
```

in the [winclient.dll] section of a client's configuration file (sql.ini) and try to access a remote database via a Windows application, *sqlspxw4.dll* is loaded into memory.

These keywords are specific to Windows SPX communications and can be configured in the [winclient.spxw4] section of the client's configuration file (sql.ini):

preferrednameservice *retry*
searchcontext *retrytimeout*

Read *Chapter 3, Configuration (sql.ini)* for complete descriptions of these keywords.

NetWare communication options

Communication libraries specify how a database server and an application communicate. On NetWare, communication between an application and a database server takes place across the network. To support remote communication, Centura provides SPX, which is a protocol-specific communication library.

You load a communication library using a *comdll* keyword in the appropriate section of the server's configuration file (sql.ini). For example:

```
[dbnwsrv.dll]
comdll=communication_library
```

The SQLBase server for NetWare (for example, *dbnwsrv.nlm*) reads the communication library section of the configuration file (for example, [dbnwsrv.dll]) when you load the database server.

You can also build a NetWare SQL/API client application residing on a NetWare client machine. Centura provides a NetWare client communication library NLM to communicate between a NetWare SQL/API application and a SQLBase server.

Multi-user database servers

The load instructions below assume that SQLBase is installed in the *c:\centura* directory. If your SQLBase location is different from *c:\centura* (for example, if you specified a different drive and directory during installation), substitute the appropriate path.

dbnservr.nlm

This is an unlimited-user SQLBase for NetWare 3.x database server. When you specify:

```
load db:\centura\dbnservr.nlm
```

at the command prompt, *dbnservr.nlm* is loaded into memory.

dbn5srvr.nlm

This is a 5-user SQLBase for NetWare 3.x database server. When you specify:

```
load db:\centura\dbn5srvr.nlm
```

at the command prompt, *dbn5srvr.nlm* is loaded into memory.

dbn10svr.nlm

This is a 10-user SQLBase for NetWare 3.x database server. When you specify:

```
load db:\centura\dbn10svr.nlm
```

at the command prompt, *dbn10svr.nlm* is loaded into memory.

dbn25svr.nlm

This is a 25-user SQLBase for NetWare 3.x database server. When you specify:

```
load db:\centura\dbn25svr
```

at the command prompt, *dbn25svr.nlm* is loaded into memory.

dbn50svr.nlm

This is a 50-user SQLBase for NetWare 3.x database server. When you specify:

```
load db:\centura\dbn50svr.nlm
```

at the command prompt, *dbn50svr.nlm* is loaded into memory.dbnwsrv.nlm

dbnwsrv.nlm

This is an unlimited-user SQLBase for NetWare 4.x database server. When you specify:

```
load db:\centura\dbnwsrv.nlm
```

at the command prompt, *dbnwsrv.nlm* is loaded into memory.

dbnw5sv.nlm

This is a 5-user SQLBase for NetWare 4.x database server. When you specify:

```
load db:\centura\dbnw5sv.nlm
```

at the command prompt, *dbnw5sv.nlm* is loaded into memory.

dbnw10sv.nlm

This is a 10-user SQLBase for NetWare 4.x database server. When you specify:

```
load db:\centura\dbnw10sv.nlm
```

at the command prompt, *dbnw10sv.nlm* is loaded into memory.

dbnw25sv.nlm

This is a 25-user SQLBase for NetWare 4.x database server. When you specify:

```
load db:\centura\dbnw25sv.nlm
```

at the command prompt, *dbnw25sv.nlm* is loaded into memory.

dbnw50sv.nlm

This is a 50-user SQLBase for NetWare 4.x database server. When you specify:

```
load db:\centura\dbnw50sv.nlm
```

at the command prompt, *dbnw50sv.nlm* is loaded into memory.

Server communication libraries

In addition to the database server executables listed in the previous section, SQLBase requires that protocol-specific server communication libraries be installed on the database server computer as well.

A SQLBase Server for NetWare supports SPX. With SPX, the server can accept connections from NetWare and MS Windows.

In the rest of this section, wherever ‘dbnwsrv’ is mentioned, it represents not only the unlimited-user version of the SQLBase Server for NetWare 4.x product, but also all user versions for both versions of NetWare. Substitute ‘dbnw5sv’, ‘dbnw10sv’, ‘dbnw25sv’, or ‘dbnw50sv’ for NetWare 4.x, or substitute ‘dbnservr’, ‘dbn5rsvr’, ‘dbn10svr’, ‘dbn25sv’, or ‘dbn50svr’ for NetWare 3.x.

spxdll40.nlm

This is a NetWare 4.x SPX communication library. When you specify:

```
comdll=sqlspx
```

in the communication library section of a server’s configuration file (sql.ini) and you load one of the multi-user database servers, *spxdll40.nlm* is loaded into memory.

The *retry* keyword is specific to NetWare 4.x SPX communications and can be configured in the SPX section of the server’s configuration file (sql.ini).

Read *Chapter 3, Configuration (sql.ini)* for a complete description of this keyword.

spxdll.nlm

This is a NetWare 3.x SPX communication library. When you specify:

```
comdll=sqlspx
```

in the communication library section of a server’s configuration file (sql.ini) and you load one of the multi-user database servers, *spxdll.nlm* is loaded into memory.

The *retry* keyword is specific to NetWare 3.x SPX communications and can be configured in the SPX section of the server’s configuration file (sql.ini).

Read *Chapter 3, Configuration (sql.ini)* for a complete description of this keyword.

Client communication libraries

SQLBase NetWare SQL/API client applications communicate through the SPX protocol. A NetWare SQL/API client application can communicate with a SQLBase Server for NetWare through SPX.

spxdll40.nlm

This is a NetWare 4.x SPX communication library. When you specify:

```
comdll=sqlspx
```

in the [nwclient.dll] communication library section of a client’s configuration file (sql.ini) and try to access a remote database via a NetWare client SQL/API application, *spxdll40.nlm* is loaded into memory.

The *searchcontext* and *preferrednameservice* keywords are specific to SPX communications and can be configured in the [nwclient.spx4] of the client’s configuration file (sql.ini).

spxdll.nlm

This is a NetWare 3.x SPX communication library. When you specify:

```
comdll=sqlspx
```

in the [nwclient.dll] communication library section of a client's configuration file (sql.ini) and try to access a remote database via a NetWare client SQL/API application, *spxdll.nlm* is loaded into memory.

The *searchcontext* and *preferrednameservice* keywords are specific to SPX communications and can be configured in the [nwclient.spx] of the client's configuration file (sql.ini).

Windows NT communication options

Using the *sqlwntm.dll* or *sqlwntw.dll* SQLAPI library, a Graphical User Interface (GUI) or character-based Windows NT client can connect to one or more local or remote database servers.

Similarly, a SQLBase for Windows NT server can support multiple client connections.

The General Communications Interface (GCI) library, *sqlngci.dll*, provides the communications interface between the client-side SQL/API library (*sqlwntm.dll* or *sqlwntw.dll*) and the lower-level communication libraries (*sqlntnbi.dll and sqlapipe.dll*). It also interfaces between the SQLBase Server for Windows NT and these same libraries.

Centura also offers a single-user Windows NT database server, which supports the same communications protocols as the multi-user server.

Multi-user database servers

The load instructions below assume that SQLBase is installed in the *c:\centura* directory. If your SQLBase location is different from *c:\centura* (for example, if you specified a different drive and directory during installation), substitute the appropriate path.

dbntsrv.exe

This is an unlimited-user SQLBase for Windows NT database server. To load the *dbntsrv.exe* into memory, select one of these options:

- Click the *dbntsrv.exe* icon, or
- Select **File**, **Run...**, then type:

  ```
  c:\centura\dbntsrv
  ```

dbnt1sv.exe

This is a 1-user SQLBase for Windows NT database server. To load the *dbnt1sv.exe* into memory, select one of these options:

- Click the *dbnt1sv.exe* icon, or
- Select **File**, **Run...**, then type:

```
c:\centura\dbnt1sv
```

dbnt5sv.exe

This is a 5-user SQLBase for Windows NT database server. To load the *dbnt5sv.exe* into memory, select one of these options:

- Click the *dbnt5sv.exe* icon, or
- Select **File**, **Run...**, then type:

```
c:\centura\dbnt5sv
```

dbnt10sv.exe

This is a 10-user SQLBase for Windows NT database server. To load the *dbnt10sv.exe* into memory, select one of these options:

- Click the *dbnt10sv.exe* icon, or
- Select **File**, **Run...**, then type:

```
c:\centura\dbnt10sv
```

dbnt25sv.exe

This is a 25-user SQLBase for Windows NT database server. To load the *dbnt25sv.exe* into memory, select one of these options:

- Click the *dbnt25sv.exe* icon, or
- Select **File**, **Run...**, then type:

```
c:\centura\dbnt25sv
```

dbnt50sv.exe

This is a 50-user SQLBase for Windows NT database server. To load the *dbnt50sv.exe* into memory, select one of these options:

- Click the *dbnt50sv.exe* icon, or
- Select **File**, **Run...**, then type:

```
c:\centura\dbnt50sv
```

Server communication libraries

In addition to the database server executables listed in the previous section, SQLBase requires that protocol-specific server communication libraries be installed on the database server computer as well.

SQLBase Server for Windows NT supports NetBIOS, Anonymous Pipes, and TCP/IP, allowing the server to connect to Netware, Windows NT, and Windows 95 clients.

sqlntnbi.dll

This communication library supports the NetBIOS protocol on a Windows NT server. When you specify:

```
comdll=sqlntnbi
```

in the [dbntsrv.dll] section of a server's configuration file (sql.ini) and start a server (for example, *dbntsrv.exe), sqlntnbi.dll* is loaded into memory.

These keywords are specific to Windows NT NetBIOS communications and can be configured in the [dbntsrv.ntnbi] section of the server's configuration file (sql.ini):

adapter	*commands*
sessions	*names*

Read *Chapter 3, Configuration (sql.ini)* for a complete description of these keywords.

sqlapipe.dll

This communication library supports the Anonymous Pipes protocol on a Windows NT server. When you specify:

```
comdll=sqlapipe
```

in the [dbntsrv.dll] section of a server's configuration file (sql.ini) and start a server (for example, *dbntsrv.exe), sqlapipe.dll* is loaded into memory.

There are no keywords specific to Anonymous Pipes on the Windows NT platform.

sqlws32.dll

This communication library supports the TCP/IP protocol on a Windows NT server. When you specify:

```
comdll=sqlws32
```

in the [dbntsrv.dll] section of a server's configuration file (sql.ini) and start a server (for example, *dbntsrv.exe), sqlws32.dll* is loaded into memory.

The *listenport* keyword is specific to TCP/IP communications and can be configured in the [dbntsrv.ws32] section of the server's configuration file (sql.ini):

Read *Chapter 3, Configuration (sql.ini)* for a complete description of this keywords

Client communication libraries

SQLBase Windows NT client software supports SPX, NetBIOS, Anonymous Pipes, and TCP/IP.

- With SPX, clients can access SQLBase for NetWare database servers.
- With NetBIOS, clients can access SQLBase for Windows NT and Windows 95 database servers.
- With Anonymous Pipes, clients can access SQLBase for Windows NT database servers.
- With TCP/IP, clients can access SQLBase for NetWare, Windows NT, and Windows 95 database servers.

sqlntnbi.dll

This communication library supports the NetBIOS protocol on a Windows NT client. When you specify:

```
comdll=sqlntnbi
```

in the [win32client.dll] section of a client's configuration file (sql.ini) and access a remote database via a Windows NT application (for example, *dbntsrv.exe*), *sqlntnbi.dll* is loaded into memory.

These keywords are specific to Windows NT NetBIOS communications and can be configured in the [win32client.ntnbi] section of the client's configuration file (sql.ini):

adapter *commands* *names* *sessions*

Read *Chapter 3, Configuration (sql.ini)* for complete descriptions of these keywords.

sqlapipe.dll

This communication library supports the Anonymous Pipes protocol on a Windows NT client. When you specify:

```
comdll=sqlapipe
```

in the [win32client.dll] section of a client's configuration file (sql.ini) and access a remote database via a Windows NT application (for example, *dbntsrv.exe*), *sqlapipe.dll* is loaded into memory.

There are no keywords specific to Windows NT Anonymous Pipes communications.

sqlws32.dll

This communication library supports the TCP/IP protocol (using Windows Sockets) on a Windows NT client. When you specify:

```
comdll=sqlws32
```

in the [win32client.dll] section of a client's configuration file (sql.ini) and access a remote database via a Windows NT application, *sqlws32.dll* is loaded into memory.

The *serverpath*, *searchcontext*, and *preferrednameservice* keywords are specific to TCP/IP communication and can be configured in the [win32client.ws32] section of the client's configuration file (sql.ini):

Read *Chapter 3, Configuration (sql.ini)* for a complete description of this keyword.

sqlwsspx.dll

This communication library supports the SPX Win32 Microsoft Client for NetWare on a Windows NT client. When you specify:

```
comdll=sqlwsspx
```

in the [win32client.dll] section of a client's configuration file (sql.ini) and access a remote database via a Windows NT application, *sqlwsspx.dll* is loaded into memory.

Note: 32-bit SPX support for Windows NT requires Microsoft's implementation of Winsock 1.1 and Microsoft Client for Netware.

The *searchcontext* and *preferrednameservice* keywords are specific to SPX communication and can be configured in the [win32client.wsspx] section of the client's configuration file (sql.ini).

sqlspx32.dll

This communication library supports the SPX Win32 Novell Client for NetWare on a Windows NT client. When you specify:

```
comdll=sqlspx32
```

in the [win32client.dll] section of a client's configuration file (sql.ini) and access a remote database via a Windows NT application, *sqlspx32.dll* is loaded into memory.

The *searchcontext* and *preferrednameservice* keywords are specific to SPX communication and can be configured in the [win32client.spx32] section of the client's configuration file (sql.ini).

Note: Currently, Novell’s Client32 for NetWare operates on Windows 95 only.

Windows 95 communication options

Using the *sqlwntm.dll* or *sqlwntw.dll* SQLAPI library, a Graphical User Interface (GUI) or character-based Windows 95 client can connect to one or more local or remote database servers.

Similarly, a SQLBase for Windows 95 server can support multiple client connections.

The General Communications Interface (GCI) library, *sqlngci.dll*, provides the communications interface between the client-side SQL/API library (*sqlwntm.dll* or *sqlwntw.dll*) and the lower-level communication libraries (*sqlntnbi.dll and sqlapipe.dll*). It also interfaces between the SQLBase Server for Windows 95 and these same libraries.

Centura also offers a single-user Windows 95 database server, which supports the same communications protocols as the multi-user server.

Multi-user database servers

The load instructions below assume that SQLBase is installed on the *c:\* drive. If your installation drive is different from *c:\*, substitute the appropriate letter drive.

dbntsrv.exe

This is an unlimited-user SQLBase for Windows 95 database server. To load the *dbntsrv.exe* into memory, select one of these options:

- Click the *dbntsrv.exe* icon (if one is installed on the desktop), or
- Select **Start**, **Run...**, then type:

```
c:\centura\dbntsrv
```

dbnt1sv.exe

This is a 1-user SQLBase for Windows 95 database server. To load the *dbnt1sv.exe* into memory, select one of these options:

- Click the *dbnt1sv.exe* icon (if one is installed on the desktop), or
- Select **Start**, **Run...**, then type:

```
c:\centura\dbnt1sv
```

dbnt5sv.exe

This is a 5-user SQLBase for Windows 95 database server. To load the *dbnt5sv.exe* into memory, select one of these options:

- Click the *dbnt5sv.exe* icon (if one is installed on the desktop), or
- Select **Start**, **Run...**, then type:

```
c:\centura\dbnt5sv
```

dbnt10sv.exe

This is a 10-user SQLBase for Windows 95 database server. To load the *dbnt10sv.exe* into memory, select one of these options:

- Click the *dbnt10sv.exe* icon (if one is installed on the desktop), or
- Select **Start**, **Run...**, then type:

```
c:\centura\dbnt10sv
```

dbnt25sv.exe

This is a 25-user SQLBase for Windows 95 database server. To load the *dbnt25sv.exe* into memory, select one of these options:

- Click the *dbnt25sv.exe* icon (if one is installed on the desktop), or
- Select **Start**, **Run...**, then type:

```
c:\centura\dbnt25sv
```

dbnt50sv.exe

This is a 50-user SQLBase for Windows 95 database server. To load the *dbnt50sv.exe* into memory, select one of these options:

- Click the *dbnt50sv.exe* icon (if one is installed on the desktop), or
- Select **Start**, **Run...**, then type:

```
c:\centura\dbnt50sv
```

Server communication libraries

In addition to the database server executables listed in the previous section, SQLBase requires that protocol-specific server communication libraries be installed on the database server computer as well.

SQLBase Server for Windows 95 supports NetBIOS, Anonymous Pipes, and TCP/IP, allowing the server to connect to Netware, Windows NT, and Windows 95 clients.

sqlntnbi.dll

This communication library supports the NetBIOS protocol on a Windows 95 server. When you specify:

```
comdll=sqlntnbi
```

in the [dbntsrv.dll] section of a server's configuration file (sql.ini) and start a server (for example, *dbntsrv.exe), sqlntnbi.dll* is loaded into memory.

These keywords are specific to Windows 95 NetBIOS communications and can be configured in the [dbntsrv.ntnbi] section of the server's configuration file (sql.ini):

adapter	*commands*
sessions	*names*

Read *Chapter 3, Configuration (sql.ini)* for a complete description of these keywords.

sqlapipe.dll

This communication library supports the Anonymous Pipes protocol on a Windows 95 server. When you specify:

```
comdll=sqlapipe
```

in the [dbntsrv.dll] section of a server's configuration file (sql.ini) and start a server (for example, *dbntsrv.exe), sqlapipe.dll* is loaded into memory.

There are no keywords specific to Anonymous Pipes on the Windows 95 platform.

sqlws32.dll

This communication library supports the TCP/IP protocol on a Windows 95 server. When you specify:

```
comdll=sqlws32
```

in the [dbntsrv.dll] section of a server's configuration file (sql.ini) and start a server (for example, *dbntsrv.exe), sqlws32.dll* is loaded into memory.

The *listenport* keyword is specific to Windows 95 NetBIOS communications and can be configured in the [dbntsrv.ntnbi] section of the server's configuration file (sql.ini):

Read *Chapter 3, Configuration (sql.ini)* for a complete description of this keyword.

Client communication libraries

SQLBase Windows 95 client software supports SPX, NetBIOS, Anonymous Pipes, and TCP/IP.

- With SPX, clients can access SQLBase for NetWare database servers.
- With NetBIOS, clients can access SQLBase for Windows NT and Windows 95 database servers.
- With Anonymous Pipes, clients can access SQLBase for Windows 95 database servers.
- With TCP/IP, clients can access SQLBase for NetWare, Windows NT, and Windows 95 database servers.

sqlntnbi.dll

This communication library supports the NetBIOS protocol on a Windows 95 client. When you specify:

```
comdll=sqlntnbi
```

in the [win32client.dll] section of a client’s configuration file (sql.ini) and access a remote database via a Windows 95 application (for example, *dbntsrv.exe*), *sqlntnbi.dll* is loaded into memory.

These keywords are specific to Windows 95 NetBIOS communications and can be configured in the [win32client.ntnbi] section of the client’s configuration file (sql.ini):

adapter	*commands*	*names*	*sessions*

Read *Chapter 3, Configuration (sql.ini)* for complete descriptions of these keywords.

sqlapipe.dll

This communication library supports the Anonymous Pipes protocol on a Windows 95 client. When you specify:

```
comdll=sqlapipe
```

in the [win32client.dll] section of a client’s configuration file (sql.ini) and access a remote database via a Windows NT application (for example, *dbntsrv.exe*), *sqlapipe.dll* is loaded into memory.

There are no keywords specific to Windows 95 Anonymous Pipes communications.

sqlws32.dll

This communication library supports the TCP/IP protocol (using Windows Sockets) on a Windows 95 client. When you specify:

```
comdll=sqlws32
```

in the [win32client.dll] section of a client's configuration file (sql.ini) and access a remote database via a Windows NT application, *sqlws32.dll* is loaded into memory.

The *serverpath*, *searchcontext*, and *preferrednameservice* keywords are specific to TCP/IP communication and can be configured in the [win32client.ws32] section of the client's configuration file (sql.ini):

Read *Chapter 3, Configuration (sql.ini)* for a complete description of this keyword.

sqlwsspx.dll

This communication library supports the SPX 32-bit Microsoft Netware protocol on a Windows 95 client. When you specify:

```
comdll=sqlwsspx
```

in the [win32client.dll] section of a client's configuration file (sql.ini) and access a remote database via a Windows NT application (for example, *dbntsrv.exe*), *sqlwsspx.dll* is loaded into memory.

The *searchcontext and preferrednameservice* keywords are specific to SPX communication and can be configured in the [win32client.ws32] section of the client's configuration file (sql.ini):

Read *Chapter 3, Configuration (sql.ini)* for a complete description of this keyword.

Note: 32-bit SPX support for Windows 95 requires that Novell's Client32 for Netware be installed on the Windows 95 client (Currently, Novell's Client32 for Netware operates on Windows 95 only, and not Windows NT).

sqlws32.dll

This communication library supports the SPX 32-bit Novell Netware protocol on a Windows 95 client. When you specify:

```
comdll=sqlws32
```

in the [win32client.dll] section of a client's configuration file (sql.ini) and access a remote database via a Windows NT application (for example, *dbntsrv.exe*), *sqlws32.dll* is loaded into memory.

The *searchcontext and preferrednameservice* keywords are specific to SPX communication and can be configured in the [win32client.ws32] section of the client's configuration file (sql.ini):

Chapter 3

Configuration (sql.ini)

This chapter describes the structure and organization of Centura’s configuration file (sql.ini) as well as the software configuration keywords for which you can specify values.

As a DBA, you need to be knowledgeable about configuration in order to tune the performance of SQLBase.

Centura configuration file

SQLBase saves configuration information in a file called sql.ini. All Centura client and server software products read this file when they start.

Finding the configuration file

If you set a value for the SQLBASE environment variable, SQLBase looks for the configuration file only in the directory you specify. Otherwise, the search order for this file is:

1. Current directory.
2. \sqlbase directory on the current drive.
3. Root directory on the current drive.
4. Directories specified by the PATH environment variable.

By default, Centura client software is installed in the Centura subdirectory and database server software is installed in the SQLBASE subdirectory.

You should have only one Centura configuration file on a computer. If you install Centura client and server software into the default directories on the same machine, or you install multiple versions of Centura software into different directories on the same machine, you end up with two configuration files. Be sure to consolidate the information into one copy of the configuration file and delete the other.

To edit the sql.ini file, use the Centura Connectivity Administrator if you are running Windows 95 or Windows NT 4.0 or later, or use your preferred text editor.

Configuration file format

The configuration file (sql.ini) is divided into sections, each starting with a section identifier enclosed in square brackets. Each section identifier corresponds to a particular product.

In most cases, section identifiers are executable names. For example, the section corresponding to SQLBase Server for Windows NT, whose executable name is *dbntsrv.exe*, is [dbntsrv].

Product section identifiers

The following tables contain lists of products and their corresponding section identifiers.

Multi-user servers

The following table lists the multi-user servers for NetWare 3.x.

Multi-user Server for NetWare 3.x	Section
SQLBase Server for NetWare 3.x (unlimited user)	[dbnservr]
SQLBase Server for NetWare 3.x (5-user)	[dbn5srvr]
SQLBase Server for NetWare 3.x (10-user)	[dbn10svr]
SQLBase Server for NetWare 3.x (25-user)	[dbn25svr]
SQLBase Server for NetWare 3.x (50-user)	[dbn50svr]

This table lists the multi-user servers for NetWare 4.x.

Multi-user Server for NetWare 4.x	Section
SQLBase Server for NetWare 4.x (unlimited user)	[dbnwsrv]
SQLBase Server for NetWare 4.x (5-user)	[dbnw5sv]
SQLBase Server for NetWare 4.x (10-user)	[dbnw10sv]
SQLBase Server for NetWare 4.x (25-user)	[dbnw25sv]
SQLBase Server for NetWare 4.x (50-user)	[dbnw50sv]

This table lists the multi-user servers for Windows NT.

Multi-user Server for Windows NT	Section
SQLBase Server for Windows NT (unlimited user)	[dbntsrv]
SQLBase Server for Windows NT (1-user)	[dbnt1sv]
SQLBase Server for Windows NT (5-user)	[dbnt5sv]
SQLBase Server for Windows NT (10-user)	[dbnt10sv]
SQLBase Server for Windows NT (25-user)	[dbnt25sv]
SQLBase Server for Windows NT (50-user)	[dbnw50sv]

This table lists the multi-user servers for Windows 95.

Multi-user Server for Windows 95	Section
SQLBase Server for Windows 95 (unlimited user)	[dbntsrv]
SQLBase Server for Windows 95 (1-user)	[dbnt1sv]
SQLBase Server for Windows 95 (5-user)	[dbnt5sv]
SQLBase Server for Windows 95 (10-user)	[dbnt10sv]
SQLBase Server for Windows 95 (25-user)	[dbnt25sv]
SQLBase Server for Windows 95 (50-user)	[dbnw50sv]

Single-user servers

Single-user Server for Windows 3.x	Section
SQLBase Windows Server (multi-tasking server)	[dbwservr]

Client communication libraries

Client Communication Libraries	Section
NetWare client application	[nwclient]
MS Windows client applications	[winclient]
MS Windows client applications (for *country* configuration keyword)	[sqlapiw]
Windows NT client applications	[win32client]
Windows 95 client applications	[win32client]

Communications

For all products except MS Windows, two additional sections also exist:

- One identifies the communications libraries (*comdlls*) that SQLBase must load to support various protocols
- The other defines values for protocol options

For example, the section corresponding to the SQLBase for Windows NT server is [dbntsrv]. The communications section for this same product is [dbntsrv.dll]. On Windows NT, SQLBase supports the following protocols:

- IBM NetBIOS
- Anonymous Pipes
- TCP/IP (using Windows Sockets)

Support for these protocols translates to the following entries in the configuration file (sql.ini):

```
[dbntsrv.dll]
comdll=sqlntnbi
comdll=sqlapipe
comdll=sqlws32
```

For each *comdll* value, a sub-section exists which defines values for protocol options. The sub-sections, each of which corresponds to a supported network protocol, are:

[dbntsrv.ntnbi](corresponds to *sqlntnbi.dll*)

[sqlapipe.dll](corresponds to *sqlapipe.dll*)

[sqlws32.dll](corresponds to *sqlws32.dll*)

Multi-user server communications

The following tables contain lists of products, their corresponding communication libraries section (where you configure one or more *comdll* keywords), and any protocol-specific sub-sections.

The table below lists communication and protocol sections for the multi-user SQLBase servers for NetWare 3.x.

# Users	Communication Libraries Section	Protocol Specific Section
Unlimited	[dbnservr.dll]	[dbnservr.spx]
5-users	[dbn5srvr.dll]	[dbn5srvr.spx]
10-users	[dbn10svr.dll]	[dbn10svr.spx]

# Users	Communication Libraries Section	Protocol Specific Section
25-users	[dbn25svr.dll]	[dbn25svr.spx]
50-users	[dbn50svr.dll]	[dbn50svr.spx]

This table lists communication and protocol sections for the multi-user SQLBase servers for NetWare 4.x.

# Users	Communication Libraries Section	Protocol Specific Section
Unlimited	[dbnwsrv.dll]	[dbnwsrv.spx]
5-users	[dbnw5sv.dll]	[dbnw5sv.spx]
10-users	[dbnw10sv.dll]	[dbnw10sv.spx]
25-users	[dbnw25sv.dll]	[dbnw25sv.spx]
50-users	[dbnw50sv.dll]	[dbnw50sv.spx]

This table lists communication and protocol sections for the multi-user SQLBase servers for Windows NT.

# Users	Communication Libraries Section	Protocol Specific Section
Unlimited	[dbntsrv.dll]	[dbntsrv.apipe] [dbntsrv.ntnbi] [dbntsrv.ws32]
1-user	[dbnt1sv.dll]	[dbnt1sv.apipe] [dbnt1sv.ntnbi] [dbnt1sv.ws32]
5-users	[dbnt5sv.dll]	[dbnt5sv.apipe] [dbnt5sv.ntnbi] [dbnt5sv.ws32]
10-users	[dbnt10sv.dll]	[dbnt10sv.apipe] [dbnt10sv.ntnbi] [dbnt10sv.ws32]

# Users	Communication Libraries Section	Protocol Specific Section
25-users	[dbnt25sv.dll]	[dbnt25sv.apipe] [dbnt25sv.ntnbi] [dbnt25sv.ws32]
50-users	[dbnt50sv.dll]	[dbnt50sv.apipe] [dbnt50sv.ntnbi] [dbnt50sv.ws32]

This table lists communication and protocol sections for the multi-user SQLBase servers for Windows 95.

# Users	Communication Libraries Section	Protocol Specific Section
Unlimited	[dbntsrv.dll]	[dbntsrv.apipe] [dbntsrv.ntnbi] [dbntsrv.ws32]
1-user	[dbnt1sv.dll]	[dbnt1sv.apipe] [dbnt1sv.ntnbi] [dbnt1sv.ws32]
5-users	[dbnt5sv.dll]	[dbnt5sv.apipe] [dbnt5sv.ntnbi] [dbnt5sv.ws32]
10-users	[dbnt10sv.dll]	[dbnt10sv.apipe] [dbnt10sv.ntnbi] [dbnt10sv.ws32]
25-users	[dbnt25sv.dll]	[dbnt25sv.apipe] [dbnt25sv.ntnbi] [dbnt25sv.ws32]
50-users	[dbnt50sv.dll]	[dbnt50sv.apipe] [dbnt50sv.ntnbi] [dbnt50sv.ws32]

Single-user server communications

Because communication between a client application and a single-user server occurs on the same computer, there is no network communication and therefore, no need to support communications protocols.

Client communications

Client	Communication Libraries Section	Protocol Specific Section
NetWare	[nwclient.cll]	[nwclient.spx]
		[nwclient.spx4]
		[nwclient.tip]
Windows (VLM) (TCP/IP)	[winclient.dll]	[winclient.nbiow]
		[winclient.spxw4]
		[winclient.wsock]
Windows NT	[win32client.dll]	[win32client.ntnbi]
		[win32client.apipe]
		[win32client.ws32]
		[win32client.wsspx]
Windows 95	[win32client.dll]	[win32client.ntnbi]
		[win32client.apipe]
		[win32client.ws32]
		[win32client.spx32]

Other

There are several sections which do not fit into either of the two cases (programs and communications) already mentioned.

Section	Description
[dbdfault]	Saves the *defaultdatabase*, *defaultuser*, and *defaultpassword* settings for Windows NT clients.
[patch]	NetFrame machine specific

Configuration entries

Each section of the configuration file (sql.ini) contains keywords and values:

```
[section identifier]
keyword1=value
keyword2=value1,value2,. . .
keyword3=value1,. . .
```

The number of values differs for each keyword.

For example:

```
[dbntsrv]
cache=300
servername=server1,sqlnpipe,sqlntnbi
dbname=demo1,sqlnpipe,sqlntnbi
```

Keywords are described later in this chapter.

Command line keywords

You can also set the value of some keywords when you start a program. Specify a keyword and its value on the command line to override a sql.ini configuration entry.

For example, the following command starts SQLBase Server for Windows NT and specifies the database home directory (*dbdir*) as *d:\test\db*. This overrides a *dbdir* configuration entry in sql.ini.

```
dbntsrv dbdir=d:\test\db
```

You can specify more than one keyword on the command line, each separated by a space, for example:

```
dbntsrv dbdir=d:\test\db log=d:\test\dblog
```

Configuration syntax rules

Use the following rules when creating configuration entries:

- Specify keywords in any order.

  For example, configuring the *servername* and *dbname* keywords in this order:

  ```
  servername=server1,sqlnpipe,sqlntnbi
  dbname=demo1,sqlnpipe,sqlntnbi
  ```

  is functionally equivalent to configuring them in this order:

  ```
  dbname=demo1,sqlnpipe,sqlntnbi
  servername=server1,sqlnpipe,sqlntnbi
  ```

- List keyword values in the order required by the keyword.

For example, you would get an error if you tried to change the order of:

```
servername=server1,sqlnpipe,sqlntnbi
```

to:

```
servername=sqlnpipe,sqlntnbi,server1
```

because the *servername* keyword expects the value of the server name to be listed first, followed by one or more communication libraries.

- Use a comma (,) to separate multiple keyword values.

  For example:

  ```
  dbname=demo1,sqlnpipe,sqlntnbi
  ```

  Note that the exception to this rule is the *dbdir* keyword which uses semicolons to separate pathnames.

- Use a semi-colon (;) to start a comment line. The rest of the line is ignored.

  For example:

  ```
  ; This specifies the name of a database as well as the
  ; protocols on which the server listens for clients ;
    making connection requests to that database.
    dbname=demo1,sqlnpipe,sqlntnbi
  ```

- Limit each entry to the length of one line.

  SQLBase would produce an error if you allowed a comment line, for example, to span multiple lines:

  ```
  ; This comment line is too long to fit on one line,
    and ends up spanning multiple lines...
  ```

- Do not place spaces before or after keyword values.

  For example, SQLBase would produce an error if you put spaces between the *dbname* keyword values:

  ```
  dbname=demo1, sqlnpipe, sqlntnbi
  ```

  You must eliminate spaces between keyword values:

  ```
  dbname=demo1,sqlnpipe,sqlntnbi
  ```

- Keywords are case-independent.

  For example:

  ```
  servername=server1,sqlnpipe,sqlntnbi
  ```

  is functionally equivalent to:

  ```
  SERVERNAME=server1,sqlnpipe,sqlntnbi
  ```

Default configuration values

If you do not specify a keyword in the configuration file (sql.ini) or on the command line, SQLBase uses a default value. The *defaults.doc* file specifies these default values.

Note: Not all of the keywords in *defaults.doc* are configurable in sql.ini.

A sample *defaults.doc:*

```
The following default values are applicable to this system.

50      CACHE, number of cache pages
30      CACHEGROUP, cache page allocation group
5       CONNECTRETRY, seconds for connect timeout
10      CONNECTPAUSETICKS, ticks for pausing
10000   HEAP, DBGATEWY heap size
145000  HEAP, DBLOCAL heap size
20000   HEAP, DBROUTER heap size
71      INTERRUPT, interrupt number
30000   NETBUFFER, size DBXROUTR network buffer
3       RETRY, number of connect retries
64      SORTCACHE, number of sort cache pages
0X8000  STACKSIZE, DBSERVER stack
0X8000  STACKSIZE, DBSIM stack
7000    STACKSIZE, DBSIM w/router stack
0       TIMEZONE
800     USERS, DBSERVER users
3       USERS, DBSIM users
1024000L checkpoint log interval in bytes
1       checkpt time interval in minutes
1000    cursor work space allocation
5       decimal precision
0       decimal scale
2000    input message buffer length
10      integer precision
20480   log buffer size in bytes
1024000Llog file size in bytes
15      maximum # of large server stacks
1000    maximum history file size
20000   maximum number of rollback log pages
100     normal file extension size
1000    output message buffer length
1024    partitioned file extension size
16000   rollback log page threshold
```

```
5       smallint precision
5       default response time out
"$$"    connect escape sequence
4       number of triggers allowed
```

Configuration keywords

The following table lists and provides a brief description of each configuration keyword:

Keyword	Brief Description
adapter	Identifies the adapter used for IBM NetBIOS communications.
cache	Specifies the number of pages in the database cache.
centurydefaultmode	Changes the two-digit default century values that SQLBase stores in a database.
characterset	Identifies a file that specifies different values for the ASCII character set.
clientcheck	Instructs the server to send the client a RECEIVE message upon receipt of a request.
clientname	Specifies the name that the server displays in the CLIENT NODE field on its Server Status display when a client connects to a database.
cmdtimeout	Specifies the amount of time to wait for a command to complete execution.
comdll	Specifies the communication libraries to load.
commands	Sets the maximum number of NetBIOS commands.
commitserver	Enables a server to act as the commit server for distributed transactions.
connecttimeout	Specifies the amount of time to wait after failing to connect to a server before attempting to connect again.
country	Instructs SQLBase to use the settings in the specified section of the *country.sql* file.
dbdir	Specifies the drives, paths, and directory names for the home database directories.

Keyword	Brief Description
dbname	Specifies a database name and, optionally, communication libraries that clients use to access the database.
dbwin	Determines the status of the Databases subwindow.
defaultdatabase	Specifies the default database name, overriding the default of DEMO.
defaultpassword	Specifies the default password, overriding the default of SYSADM.
defaultuser	Specifies the default user name, overriding the default of SYSADM.
defaultwrite	Controls whether changes to the values of the *defaultdatabase*, *defaultpassword*, or *defaultuser* keywords are written to the configuration file.
directsap	Allows you to specify a Service Advertising Protocol (SAP) engine for SQLBase servers running under NetWare 4.x.
disablelogspacecheck	Allows you disable checking for log file space availability before a new log file is opened.
displevel	Specifies the level of information (0-4) displayed on the Process Activity window.
errorfile	Specifies a file that contains entries to translate standard SQLBase return codes into user-defined return codes.
extdll	Specifies the pre-loading of DLLs at server startup time.
fileaccess	Prevents user access to select SQL/API functions.
groupcommit	Specifies the maximum number of *commits* that SQLBase groups together before physically writing them to disk.
groupcommitdelay	Specifies the maximum number of system ticks that SQLBase waits before performing a *commit*.
ibmnetbios	Identifies the adapter used for IBM NetBIOS communications.
inmessage	Sets the size of the input message buffer.
insertioncontext	Specifies where in the NDS Tree SQLBase server and database objects reside.

Keyword	Brief Description
listenport	Identifies the port number on which a server listens for connections.
listenretry	Specifies the number of times a server should retry a *listen* when an attempt to listen on the network fails.
locks	Specifies the maximum number of lock entries to allocate.
locktimeout	Specifies how long to wait when acquiring a lock.
log	Writes all the messages that appear on the server's Process Activity display to a specified file.
logdir	Redirects the transaction logs to the specified drive and directory.
logfileprealloc	Enables and disables transaction log file preallocation.
mainwin	Determines the size and position of the main server window.
maxnestinglevel	Specifies the maximum nesting level for recursing stored procedures.
names	Sets the number of installable NetBIOS names.
ndsloginid	Specifies the userid to log into the NDS Tree.
ndsloginpassword	Sets a password for the NDS user ID that is used to log into the NDS Tree.
netcheck	Enables and disables a checksum feature that detects transmission errors between a client and a server.
netchecktype	Specifies the algorithm SQLBase uses when *netcheck* is enabled.
netlog	Invokes a diagnostic server utility that records database messages to a specified log file.
numconnectionecb	Adjusts the number of connection Event Control Blocks (ECBs).
numlistenecb	Used to adjust the number of listen Event Control Blocks (ECBs).
nwadvertisemode	Specifies what mode to use when advertising SQLBase servers and databases on the network.

Keyword	Brief Description
optimizefirstfetch	Sets the optimization mode for a result set.
optimizerlevel	Determines the optimizing techniques that SQLBase uses for all clients that connect to a server.
oracleouterjoin	Turns on and off Oracle-style outer join processing.
osavgwindow	Sets the number of samples of the operating system statistics to keep for determining average value.
ossamplerate	Sets the frequency at which operating system statistics are gathered.
outmessage	Sets the size of the output message buffer.
password	Sets a password for the server.
patch	NetFrame machine-specific.
preferrednameservice	Specifies a Novell name service to use.
procwin	Determine the status of the Process Activity subwindow.
readonly	Enables and disables read-only mode for all databases on a server.
retry	Specifies the number of times a DOS or Windows NetBIOS client or a Windows client should attempt to connect to a server.
retrytimeout	Specifies how long a Windows client using either the NetBIOS or SPX protocol should wait for a response from a server when attempting to connect.
searchcontext	Specifies what part of the NDS Tree that an NDS client requires to search for the SQLBase server name and installed database names.
servername	Specifies a name for a multi-user server.
servernames	Specifies the names of one or more LAN Manager file servers to which a database server connects.
serverpath	Indicates the path that a client using the TCP/IP protocol uses to locate SQLBase servers.
serverprefix	Lets you change the character that SQLBase uses as the prefix for server names.

Keyword	Brief Description
sessions	Sets the maximum number of sessions that can be allocated for a process.
showmaindbname	Enables or disables the display of the MAIN database when you query the server for a list of database names.
silent	Turns the display for a multi-user server on and off.
sortcache	Specifies the number of cache pages to use for sorting.
statwin	Determines the status of the Server Status subwindow.
syswin	Determines the status of the System Activity subwindow.
tempdir	Specifies the directory where SQLBase places temporary files.
threadmode	Specifies whether to use native NetWare threads or SQLBase threads.
threadstacksize	Specifies the stack size.
timecolononly	Configures SQLBase to recognize when a delimiter other than a colon (:) is being used to separate the hours, minutes, and seconds portions of a time value.
timeout	Specifies the time period that the server waits for a client to make a request.
timestamps	Specifies the timestamp setting.
timezone	Sets the value of SYSTIMEZONE, a SQLBase keyword that returns the time zone as an interval of Greenwich Mean Time.
users	Specifies the maximum number of client applications (or processes) that can connect to a server simultaneously.
workalloc	Specifies the basic allocation unit of a work space.
worklimit	Specifies a maximum memory limitation for SQL commands.

The detailed configuration keyword descriptions that follow contain:

- Purpose
- Default value, if any
- Example
- Whether configuration is mandatory, recommended, or optional

- Where to configure the keyword within the configuration file (sql.ini)

If you incorrectly configure a keyword, SQLBase returns an error when you try to start the appropriate program.

adapter

Purpose. Identifies the adapter used for IBM NetBIOS communications. This keyword is passed down to NetBIOS. This keyword is not valid for NetWare servers.

Default value. This keyword has no default value; specify 0 or 1 to refer to the first or second network adapter, respectively.

Example. To identify the second adapter as that used for IBM NetBIOS communications, specify:

```
adapter=0
```

Mandatory/Unnecessary. Configuration of this keyword is mandatory if you are using IBM NetBIOS; otherwise, it is unnecessary.

Where configured. Configure this keyword in the following section of the configuration file (sql.ini):

- Windows client: [winclient.nbiow] if *ibmnetbios=1* is also set

audit

Purpose. Specifies the path and directory name for the audit file managed in the sql.ini file by the START AUDIT and STOP AUDIT commands. For details on database auditing, read the *Database Administrator's Guide*. For descriptions of the parameter entries that are automatically provided when you start an audit, read the START AUDIT command documentation in the *SQL Language Reference*.

cache

Purpose. Specifies the number of pages in the database cache. The cache buffers database pages in memory. The larger the cache, the less the disk input and output. In other words, as you increase the value of the *cache* setting, disk access is reduced.

Note that the combined values of *cache* and *sortcache* cannot exceed the memory resources of your system.

Default value. The default cache size is:

- Windows: 500K
- All other platforms: 2M

The minimum is 15K, and the maximum is 32767K.

You should experiment with various *cache* values to determine the best setting. The best setting is that which maximizes the performance of your applications.

Example. To set the number of pages in cache to 200, specify:

```
cache=200
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

centurydefaultmode

Purpose. By default, SQLBase always stores 2-digit century values as the current century. To change the default setting, specify 1 (one) as the value for the *centurydefaultmode* keyword. When set to 1, SQLBase applies the algorithm reflected in the following table to determine whether the year is in the current, previous, or, next century.

When last 2-digits of current year are:	When 2-digit entry is 0-49	When 2-digit entry is 50-99
0-49	The input date is in the **current** century	The input date is in the **previous** century
50-99	The input date is in the **next** century	The input date is in the **current** century

Note: Enabling the 2-digit century is a SQLBase feature and has no impact on connectivity routers. If you are using a Centura developed application or a SQL/API application against a non-SQLBase database, read the database documentation for information on how it determines year/century values.

Example. Assume the current year is 1996:

If 05 is entered, the computed date is 2005
If 89 is entered, the computed date is 1989

- Assume current year is 2014:

  If 05 is entered, the computed date is 2005
  If 34 is entered, the computed date is 2034
  If 97 is entered, the computed date is 1997

- Assume current year is 2065:

  If 05 is entered, the computed date is 2105
  If 70 is entered, the computed date is 2070

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

characterset

Purpose. Identifies a file that specifies different values for the ASCII character set.

This is useful for non-English speaking countries where characters in the ASCII character set have different hexadecimal values than those same characters in the U.S. ASCII character set. For example, in the U.S., 0x41 is an 'A'. In another country, 0x41 might be a 'G'.

When you insert data into a database, the specified file is used by SQLBase to translate non-U.S. ASCII characters to U.S. ASCII characters.

Example. For example, assume you specify:

```
characterset=abc.chr
```

and the file *abc.chr* contains the line:

```
0x09, 0x10, 0x45, ...
```

SQLBase makes the following translation:

From...To...

0x090x00
0x100x01
0x450x02
...

SQLBase reverses the process when you fetch the data back, making the following translation:

From...To...

0x000x09
0x010x10
0x020x45
...

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

clientcheck

Purpose. Tells SQLBase to send the client a RECEIVE message upon receipt of a request.

Default value. By default, *clientcheck* is off (0). When SQLBase has finished executing a command, it issues a SEND request to the client with the results of the command. If the SEND message is successful, the server then issues a RECEIVE request and waits to receive another command.

Setting *clientcheck* on (1) instructs SQLBase to issue a RECEIVE request before beginning execution of the command, not after it finishes executing the command. Doing so allows SQLBase to detect a situation where the client session is dropped or a cancel request is made during command processing.

Example. To set *clientcheck* on, specify:

```
clientcheck=1
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

clientname

Purpose. Specifies the name that a server displays in the CLIENT NODE field on its Server Status display when a client connects to a database.

Default value. Based on the platform, SQLBase has various ways of generating the CLIENT NODE value displayed on the Server Status screen:

- For Windows CLIENT NODE displays a randomly-generated number.
- For a client whose configuration file (sql.ini) includes the *timebased* configuration keyword, CLIENT NODE displays a timestamp.

Specifying a value for the *clientname* keyword overrides this default. The value of *clientname* can be from 1 to 12 characters in length.

Example. To name a client 'workstation1', specify:

```
clientname=workstation1
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in any client communication library section of the configuration file (sql.ini) except [sqlos2].

cmdtimeout

Purpose. Specifies the amount of time (in seconds) the server should wait for a SELECT, INSERT, UPDATE, or DELETE command to complete execution. After the specified time has elapsed, SQLBase rolls back the command.

Default value. Valid values are:

1 to 43,200 (1 second to 12 hours)

0 (infinity; wait forever; default)

Example. To set a limit of one minute, specify:

```
cmdtimeout=60
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

comdll

Purpose. Specifies the communication libraries (DLLs) to load. Communication libraries specify the mechanism that provides communications between a client and a server. Communication can be across a network or interprocess on a single computer.

For all products (except Windows server products), there is a communication library that needs to be loaded for each protocol supported.

Platform	Protocol	Comdll value
NetWare	SPX (3.x)	sqlspx
	SPX (4.x)	sqlspx
	TCP/IP (through TLI)	sqltip
Windows	Local multi-tasking server	sqlwsv
	NetBIOS	sqlnbiow
	SPX (Novell’s NetX shell)[1]	sqlspxw
	SPX (Novell’s DOS/VLM)	sqlspxw4
	TCP/IP	sqlwsock

Platform	Protocol	Comdll value
Windows NT	NetBIOS	sqlntnbi
	TCP/IP	sqlws32
	SPX (32-bit)[2]	sqlwsspx
	Anonymous Pipes[3]	sqlapipe
Windows 95	NetBIOS	sqlntnbi
	TCP/IP	sqlws32
	SPX (32-bit)[2]	sqlwsspx
	SPX (32-bit)[4]	sqlspx32
	Anonymous Pipes[3]	sqlapipe

[1] Novell's Windows 3.1 NETX client router is provided in this release, but not supported.

[2] Note that 32-bit SPX support for Windows 95 and Windows NT requires Microsoft's implementation of Winsock 1.1 and Microsoft Client for NetWare. Due to a Microsoft bug in the Win95 Winsockets implementation, this release of SQLBase does not support Winsockets on the Windows 95 platform.

[3] Note that Anonymous Pipes is local-only for Windows 95 and Windows NT.

[4] Note that 32-bit SPX support for Windows 95 requires Novell's Client32 for NetWare be installed on the Windows 95 client (Currently, Novell's Client32 for NetWare operates on Windows 95 only, and not Windows NT).

Example. For example, if a Windows NT client and server communicate via NetBIOS, specify:

```
comdll=sqlntnbi
```

This tells the server to listen for the client's connection request on the Windows NT NetBIOS protocol and tells the client to use this same protocol to make the connection request.

You can specify one or more communication libraries in a client or server's configuration file (sql.ini). A server can listen on multiple protocols simultaneously. A client can only use one protocol at a time, so a client attempts to make a connection with each communication library in turn until it finds one that works.

For example, if your network supports both NetBIOS and SPX but a client uses NetBIOS more often, the *comdll* keyword entries should be in this order:

```
[win32client.dll]
comdll=sqlntnbi
comdll=sqlwsspx
```

This ensures that SQLBase first tries to make the connection via Windows NT NetBIOS, and only if that fails does it try to make the connection via SPX.

Mandatory. Configuration of this keyword is mandatory.

Where configured. Configure this keyword in the server communications or client router communications section of the configuration file (sql.ini).

commands

Purpose. Sets the maximum number of NetBIOS commands.

During a network session, NetBIOS commands are issued. Some commands execute in non-blocking mode (also called no-wait mode). When a NetBIOS command executes in non-blocking mode, the program does not wait until the command completes. Before the command completes, another NetBIOS command can be issued. This is why a single NetBIOS session can use more than one command.

The SQLBase server itself uses one command, and you need to allocate a command for each database on which the server is listening. As well, each connected client process uses at least one command and no more than two commands. For example, a process uses an extra command when an application switches between cursors. Therefore, set *commands* higher than *sessions*. In general, the following is a good rule of thumb:

```
commands = sessions + 20%
```

Default value. This keyword has no default value.

Example. To set commands to 64, specify:

```
commands=64
```

Optional. Configuration of this keyword is optional.

Note: The *sessions, commands,* and *names* keyword values cannot exceed the values defined in the IBM configuration file. Reconfiguring these keywords can require adjusting both the IBM network configuration file and Centura's configuration file.

Where configured. Configure this keyword in a Windows NT multi-user server's NetBIOS communications ([dbntsrv.ntnbi]) section of the configuration file (sql.ini) or in a Windows 95 multi-server's NETBIOS communications (dbntsrv.ntnbi]) section of the configuration file.

Where configured. Configure this keyword in a multi-user server's NETBIOS communications section of the configuration file (sql.ini). For the Windows NT and Windows 95 platforms, this is [dbntsrv.ntnbi].

commitserver

Purpose. Enables a SQLBase server to act as the commit server for distributed transactions.

The two-phase commit protocol uses a commit server that logs information about each distributed transaction. The commit server performs the following functions:

- Assigns a global ID to the transaction. This global ID distinguishes a distributed transaction from a regular transaction.
- Starts and ends a distributed transaction.
- Commits or rolls back a distributed transaction.
- Logs the status and various stages of a distributed transaction.
- Assists in recovery in case of a crash.

One of the servers participating in the distributed transaction must have *commitserver* enabled. If both servers are enabled, SQLBase arbitrarily selects one to be the commit server.

Default value. By default, *commitserver* is not enabled (0).

Example. To enable commit server capability, specify:

```
commitserver=1
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in any multi-user server section of the configuration file (sql.ini).

connecttimeout

Purpose. Specifies the amount of time (in seconds) that a client should continue to try to connect to a server. Extra time may be needed, for example, in a case where you have just created a database and the network has to post that newly-created database before you can connect to it.

Default value. By default, *connecttimeout* is 5. The minimum value is 1 and the maximum value is 32,000.

Example. To set *connecttimeout* to 20, specify:

```
connecttimeout=20
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in a Windows client's [winclient] section.

country

Purpose. Instructs SQLBase to use the settings in the specified section of the *country.sql* file.

Default value. SQLBase supports English as the default language.

Language. SQLBase also supports many international languages including those spoken in Europe and Asia. You specify information that enables SQLBase to support another language in the *country.sql* file. This file contains a section for each country that SQLBase supports, and the section name is the country name as coded for the *country* keyword in sql.ini.

To support French:

```
country=france
```

The matching entry in the *country.sql* file would look like this:

```
[france]
keyword1=value
keyword2=value
```

ASCII versus EBCDIC. The *country* keyword also enables you to choose between using ASCII or EBCDIC formatted data on your server machine. Using EBCDIC mode enhances the testing of programs designed to run on a mainframe, and eases data exchange between the mainframe and the PC.

The *country* keyword corresponds to an entry in the *country.sql* file where you can configure an entry in the [ebcdic] section to define the EBCDIC sort sequence.

Example. To support German, specify:

```
country=germany
```

Optional. Configuration of this keyword is optional.

Where configured. Configure in any of the following configuration file (sql.ini) sections:

- A Windows client: [sqlapiw]
- Any client router or server section

For more information, read *Chapter 11, National Language Support*.

dbdir

Purpose. Specifies the drives, paths, and directory names for the home database directories. For partitioned databases, *dbdir* indicates the location of the MAIN database when you create the first partitioned database.

A home database directory contains a subdirectory which in turn contains a database file and, optionally, one or more log files. The name of the subdirectory must be the same as the database file name without the extension. For example, the home database directory for the *demo.dbs* database is the directory that contains the DEMO subdirectory.

Note that the home database directories are specified with a drive letter and path name. If you omit the drive letter, the current drive is assumed.

If you set a value for the SQLBASE environment variable, SQLBase looks for a database only in the directory you specify. Otherwise, the search order for the database file is:

1. Current directory.
2. *\sqlbase* directory on the current drive.
3. Root directory on the current drive.
4. Directories specified by the PATH environment variable.

SQLBase creates a new database in the first directory on the *dbdir* path or in the first directory on the search paths above if *dbdir* is not specified.

If the configuration file contains two *dbdir* keywords, the second setting overrides the first.

The SQLTalk SET DBDIR command and the *sqlset* API function (with the SQLPDBD parameter) change the setting of *dbdir*.

Default value. By default, CENTURA is the home database directory.

Example. You can specify a path with one or more directories. The maximum length of the *dbdir* value is 260 characters. The maximum number of characters allowed in a *dbdir* directory path (excluding the filename which has a limit of eight characters and a three-character extension) is 246. This includes the drive letter, colon, leading backslash, and terminating null character.

Semicolons and commas cannot be part of any pathname. Semicolons are used are path delimiters, while commas are used as special characters to separate each parameter of a keyword in sql.ini. A directory name enclosed in single quotes can contain an embedded single quote which is indicated by TWO single quotes (for example: D:\'"test"'\

The format is the same as for the DOS path command:

```
dbdir=d:\centura;d:\devel;d:\test
```

Note: Characters reserved in the Windows environment cannot be used in directory or file names; for example: **<> : “ / \ |**. Words reserved in the Windows environment cannot be used as filenames; for example, *con* or *aux*.

For SQLBase Server for NetWare, be sure to include the volume (db:):

```
dbdir=db:\centura
```

Recommended. Configuration of this keyword is recommended.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

dbname

This keyword has slightly different uses depending on whether you specify it in the server’s or the client’s configuration file (sql.ini).

Client configuration

Purpose. Specifies a database name and communications libraries that the client should use to connect to that database. The client can connect to the database using *only* those communications libraries.

If you do not specify a *dbname* keyword, the client tries to connect to a database using the communications libraries specified by the *comdll* keyword in the [application.dll] section, in the order in which you listed the *comdll* keywords.

Default value. The value of this keyword defaults to `demo`.

Example. To connect to the test database using IBM NetBIOS, specify:

```
dbname=test,sqlibmn2
```

You can also specify a wildcard character (*) for the database name:

```
[application]
dbname=*,sqlibmn2
```

The wildcard matches any database name that the user specifies. The above example directs the client to connect to the database using the IBM NetBIOS protocol.

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

Server configuration

Purpose. Specifies a database name on the server. This name must be unique on the network. The server listens on the network for client requests to access this database and when a client attempts to connect to the database, the server helps establish the connection.

Make sure that for each *dbname* entry, there exists a corresponding database. Otherwise, the network does not recognize the database.

The following SQL commands and SQL/API functions update the *dbname* setting:

SQL Command	SQL/API Function
CREATE DATABASE	*sqlcre*
DROP DATABASE	*sqldel*
INSTALL DATABASE	*sqlind*
DEINSTALL DATABASE	*sqlded*

Default value. The value of this keyword defaults to demo.

Example. To identify the test database, specify:

```
dbname=test
```

Remember that there must be a *test.dbs* file in the TEST subdirectory of the home database directory or in the path defined by the database home directory *dbdir*.

You can optionally specify one or more communication library names after the database name. For example, SQLBase Server for Windows NT supports NetBIOS, Anonymous Pipes, SPX (Microsoft Client for NetWare), and SPX (Novell NetWare Client). The following configuration entry directs the server to listen on the network for connection requests to the database using *all* of those communication libraries:

```
dbname=test,sqlntnbi,sqlapipe,sqlws32,sqlwsspx
```

Conversely, the following configuration entry directs the server to listen on the network using *only* the NetBIOS communication library:

```
dbname=test,sqlntnbi
```

Only those client requests to access the test database that are sent via NetBIOS will be successful.

Remember that any communication libraries that you specify for the *dbname* configuration entry, must also be specified for the *comdll* configuration entry.

Recommended. Configuration of this keyword is recommended.

Where configured. Configure this keyword in any multi-user server section of the configuration file (sql.ini).

dbwin

Purpose. Determines the status of the Databases subwindow.

The syntax for this window keywords is identical to that for the ***mainwin*** keyword, with the addition of a ***closed*** status. This indicates that the given subwindow was closed when you last used the **Save** option of the **File** menu.

The format of this keyword is:

```
dbwin=status,x,y,cx,cy
```

Optional. Configuration of this keyword is optional.

Where configured. This keyword is configured automatically in the server's GUI section (for example, [dbntsrv.gui]) when you use the **Save Settings** option in the server's **File** menu. Do not set a value for this keyword manually.

defaultdatabase

Purpose. Specifies the default database name, overriding the default of DEMO.

Default value. This keyword has no default value.

Example. To set the default database name to 'test', specify:

```
defaultdatabase=test
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the Windows NT default-specific section [dbdfault], or in a Windows client's [winclient] section.

defaultpassword

Purpose. Specifies the default password, overriding the default of SYSADM.

Default value. This keyword has no default value.

Example. To set the default password to 'secret', specify:

```
defaultpassword=secret
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the Windows NT default-specific section [dbdfault] or in a Windows client's [winclient] section.

defaultuser

Purpose. Specifies the default user name, overriding the default of SYSADM.

Default value. This keyword has no default value.

Example. To set the default user name to 'admin', specify:

```
defaultuser=admin
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the Windows NT default-specific section [dbdfault] or in a Windows client's [winclient] section.

defaultwrite

Purpose. Controls whether changes to the values of the *defaultdatabase*, *defaultpassword*, or *defaultuser* keywords are written to the configuration file.

Default value. Valid values are 0 (off) and 1 (on). The default is 1.

Example. To set *defaultwrite* off, specify:

```
defaultwrite=0
```

This specifies that changes to the *defaultdatabase*, *defaultpassword*, and *defaultuser* keywords are in effect for the current session only and are not written to the configuration file.

When *defaultwrite* is on, changes to the default keywords are remembered by SQLBase for the current, and all future, sessions. As well, the specified changes are written to the configuration file.

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the Windows NT default-specific section [dbdfault] or in a Windows client's [winclient] section.

directsap

Purpose. Allows you to specify whether to use Novell’s Service Advertising Protocol (SAP) engine or the alternative Centura SAP engine for the SQLBase Server for NetWare 4.x.

Under NetWare 4.x, the Novell SAP engine has a maximum of 20 SAPs. This means that if you have more than 15 databases installed and you try to connect to additional databases, the connection fails. Using the Centura SAP engine removes this limitation.

The syntax of this keyword is:

```
directsap=n
```

where n is either 0 (use Novell’s SAP engine) or 1 (use Centura’s SAP engine).

Example. To use Centura’s SAP engine, specify:

```
directsap=1
```

Default value. The default is 0.

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the SPX section for the SQLBase Server for NetWare 4.x, for example, [dbnwsrv.spx] for the unlimited version). This keyword is not applicable to the SQLBase Server for NetWare 3.x, since there is no SAP limit under NetWare 3.x.

disablelogspacecheck

Purpose. Disables SQLBase from checking log file space availability before a new log file is opened. SQLBase opens a new log file when the current one has reached its specified maximum size. If the log file space is inadequate to ensure database integrity should a system failure occur, SQLBase selectively rolls back any transaction that is causing excessive logging. SQLBase issues an error message that informs you the transaction has been rolled back, asks you to check for free disk space, and warns of imminent system shut down if the roll back does not recover log disk space.

By default, LOBACKUP is off, so that SQLBase deletes log files without backups as soon as they are not needed to perform transaction rollback or crash recovery. This prevents log files from accumulating and filling up the disk. When *disablelogspacecheck* is enabled, be sure to set LOGBACKUP to off; otherwise, SQLBase is unable to attempt recovery of log disk space. To set LOGBACKUP, use the SQLTalk SET LOGBACKUP command or the *sqlset* API function.

Example. The syntax is:

```
disablelogspacecheck=n
```

Default value. When this keyword is not declared, the default behavior is to perform checking for available log file space. Otherwise, to specify the default behavior, the value is 0. To disable checking, the value is 1.

Example. To disable checking for available log file space, specify:

```
disablelogspacecheck=1
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini). This keyword is not applicable to Windows 3.x and on any partitioned database.

displevel

Purpose. Specifies the level of information (0-4) displayed on the Process Activity window. Set the display level with either the Level menu on the GUI server, the SQLTalk SET PRINTLEVEL command, or the ***sqlset*** SQL/API call in conjunction with the SQLPPLV parameter.

Example. The syntax is:

```
displevel=n
```

Optional. Configuration of this keyword is optional.

Where configured. This keyword is configured automatically in the server's GUI section (for example, [dbntsrv.gui]) when you use the **Save Settings** option in the server's **File** menu. Do not set a value for this keyword manually.

errorfile

Purpose. Specifies a file that contains entries to translate standard SQLBase return codes into user-defined return codes.

Default value. This keyword does not have a default value.

Example. The syntax is:

errorfile=*filename*

where *filename* includes the full path.

The maximum length of the *errorfile* value is 260 characters. The maximum number of characters allowed in an *errorfile* directory path (excluding the filename which has a limit of eight characters and a three-character extension) is 246. This includes the drive letter, colon, leading backslash, and terminating null character.

Semicolons and commas cannot be part of the pathname. A directory name enclosed in single quotes can contain an embedded single quote which is indicated by TWO single quotes (for example: D:\'"test"'\).

Note: Characters reserved in the Windows environment cannot be used in directory or file names; for example: <> **:** **“** / \ |. Words reserved in the Windows environment cannot be used as filenames; for example, *con* or *aux*.

The file contains entries for error code translation in the form:

```
sbrcd,udrcd
```

where *sbrcd* is a SQLBase return code found in *error.sql*, and *udrcd* is a user-defined return code. The *sbrcd* value must be a positive integer; the *udrcd* can be a positive or negative integer. There can be no white space between the values or after the comma.

The client application converts the *sbrcd* value to the *udrcd* value using the *sqltec* API function. For example, SQLBase returns a value of '1' to indicate an end-of-fetch condition, while DB2 returns a value of '100'. If you want an application to convert all SQLBase return codes of '1' to '100', the entry in the errorfile would look like this:

```
1,100
```

When your application calls the *sqltec* function, if the SQLBase return code doesn't exist, SQLBase returns a non-zero return code that means that the translation did not occur.

To force translation to occur, you can create a global translation entry using the asterisk (*) character and a generic return code (like '999'). For example, assume we created an errorfile of SQLBase return codes and corresponding DB2 return codes. For those SQLBase return codes that have no corresponding DB2 return code, you can force the application to return the generic return code '999' with the following entry:

```
*,999
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the Windows NT default-specific section [dbdfault], or in a Windows client's [winclient] section.

extdll

Purpose. Specifies the pre-loading of DLLs at server startup time. Because of the enormous overhead involved in making calls to the Microsoft API function load library, (especially in the case of large DLLs or DLLs that cause more DLLs to be loaded), pre-loading DLLs saves overhead and guarantees that the DLL is loaded as long as the server is running.

Note: If you have a DLL that uses global variables that can be accessed from different functions or from multiple invocations of a function, you must load the DLL at server start up.

If you want to pre-load more than one DLL, you can specify the parameter multiple times within the server section. Each DLL entry must be on a separate line.

Default value. By default, SQLBase loads the DLL at function call time by calling the Microsoft API function Load Library.

Example. The syntax is:

```
EXTDLL=dllname
```

where *dllname* is a full path or the Operating System uses the path environment variable to locate *dllname*. If the DLL name is not qualified, the Operating System uses the path environment variable to locate the DLL.

The maximum length of the *extdll* value is 260 characters. The maximum number of characters allowed in the *extdll* directory path (excluding the filename which has a limit of eight characters and a three-character extension) is 246. This includes the drive letter, colon, leading backslash, and terminating null character.

Semicolons and commas cannot be part of any pathname. Semicolons are used as path delimiters, while commas are used as special characters to separate each parameter of a keyword in sql.ini. A directory name enclosed in single quotes can contain an embedded single quote which is indicated by TWO single quotes (for example: D:\'"test"'\.

Note: Characters reserved in the Windows environment cannot be used in directory or file names; for example: <> : “ / \ |. Words reserved in the Windows environment cannot be used as filenames; for example, *con* or *aux*.

Optional. Configuration of this keyword is optional.

Where configured. Add the EXTDLL=*dllname* keyword to the dbwservr or dbntsrv (whichever applies to your environment) server section of the sql.ini file.

fileaccess

Purpose. Allows or disallows a user ability to access and modify files on a remote server machine. Use the *fileaccess* keyword with any function that allows remote access to any type of remote file.

Default value. The default for this keyword is compatible with the existing behavior of database servers.

Example. The syntax is:

```
sqlmwr
fileaccess=0
```

Where *sqlmwr* is a function for writing to a remote server file. Setting *fileaccess*=0 disallows the user to write to a remote server file. (*fileaccess*=1 allows the user to perform the function.)

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the sql.ini file on the server machine in association with a file access function.

groupcommit

Purpose. Specifies the maximum number of COMMITs that SQLBase groups together before physically writing them to disk. *Commits* are performed when:

- The number of grouped *commits* exceeds the value of *groupcommit*, or
- The number of ticks that have elapsed since the last *commit* is greater than the value of *groupcommitdelay*. Read the *groupcommitdelay* description in the next section for more information on the *groupcommitdelay* keyword.

SQLBase tries to economize resources and increase transaction rates by grouping COMMITs from several transactions into one transaction log entry, and performing them all in one physical write. It does this by pausing for a fraction of a moment in anticipation of other incoming COMMIT requests. SQLBase does not defer COMMITs; it always writes a COMMIT to disk before returning control to the user.

Default value. The default value is 1.

Example. To set the maximum to 5, specify:

```
groupcommit=5
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

groupcommitdelay

Purpose. Specifies the maximum number of system ticks that SQLBase waits before performing *commits*. The duration of a system tick is platform-dependent, but is approximately 1/20 of a second.

Commits are performed when:

- The number of grouped *commits* exceeds the value of *groupcommit*, or
- The number of ticks that have elapsed since the last *commit* is greater than the value of *groupcommitdelay*.

Read the *groupcommit* description in the previous section for more information on the *groupcommit* keyword.

Default value. The default value is 1 tick.

Example. To set the groupcommitdelay to 20, specify:

```
groupcommitdelay=20
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

inmessage

Purpose. Sets the size (in bytes) of the input message buffer. The input message buffer holds input to the application (such as the result of a query).

There is one input message buffer per connected cursor on the client computer. The database server maintains a buffer that is the size of the largest input message buffer on the client computer. The server builds a message in its buffer, then sends it across the network to the client's input message buffer. It is called an input message buffer because it is input from the client's perspective.

Default value. By default, SQLBase maintains an input message buffer large enough to hold at least one row of data. Despite the *inmessage* value, SQLBase dynamically allocates more space if necessary.

When fetching data, SQLBase compacts as many rows as possible into the input message buffer. Each fetch reads the next row from the buffer until all have been read. At this point, SQLBase transparently fetches another buffer- ful of rows, depending on the isolation level.

A large input message buffer can improve performance when reading long varchar data and while fetching data because it reduces the number of network messages between the client and server. A large buffer can have a negative impact on concurrency, however, because any row currently in the buffer can have a shared lock on it (depending on the isolation level) which prevents other users from changing that row. Read the *sqlsil* documentation in the *SQLBase Application Programming Interface Reference* for more information about how each isolation level uses the input message buffer.

Example. To set the size of the input message buffer to 5,000 bytes, specify:

```
inmessage=5000
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in any client router section of the configuration file (sql.ini), except for the Windows NT platform.

insertioncontext

Purpose. Allows you to specify where in the NDS Tree the SQLBase server and database objects are to reside.

The syntax for this keyword is:

insertioncontext=*object_name*

where *object_name* can be a complete or partial object name. If you use a complete object name you provide the path from the root of the NDS Tree to the particular object. Standard abbreviations for commonly used name types are:

Object type	Abbreviation
Country	C
Organization Name	O
Organizational Unit	OU
Name	N
Locality Name	L
Common Name	CN
State or Province Name	S
Street Address	SA

Note: If you are specifying a search context for client users, be sure the *searchcontext* keyword in the client sections of sql.ini are set to the same value as the *insertioncontext* keyword.

Example. To specify that the SQLBase server and database objects reside in a part of the Directory Tree identified as CN=SERVERxx, you enter:

```
[dbnwsrv]
insertioncontext=CN=SERVERxx.OU=SQLBASE.OU=Menlo.O=Centura
```

Note that each path entry is separated by a period. No period is required if you are providing only one path entry, or if you are terminating the path entries. You may also provide a partial object name.

Default value. If you do not specify this keyword, the default context of the user specified by the *ndsloginid* keyword is used for insertion.

Optional. Configuration of this keyword is optional.

Where configured. For SQLBase Server for NetWare 4.x, configure this keyword in the server section of the configuration file (sql.ini).T his keyword is not applicable to the SQLBase Server for NetWare 3.x.

listenport

Purpose. Identifies the port number on which a server listens for connections. On a given machine, a SQLBase server obtains exclusive use of the identified port. You must ensure that the port chosen does not conflict with other applications on the same machine.

If you specify this keyword, client applications connecting to this server must use the same port number with the *serverpath* keyword in their corresponding client section. If you specify listenport in a client section, you will receive an error.

Contact your network administrator if you do not know what port value to use for this keyword.

Default value. The default port number is 2155.

The format of this keyword is:

```
listenport=n
```

where *n* is a positive integer not greater than 32767.

Example. To set the port Number to 3000, enter the following line:

```
listenport=3000
```

Optional. This keyword is optional.

Where configured. Configure this keyword in a server’s TCP/IP section of the sql.ini file. For the Windows NT and Windows 95 platforms, this is [*servername*.ws32], for example, [dbntsrv.ws32]. For the NetWare platform, this is [*servername*.tip], for example, [dbnservr.tip].

listenretry

Purpose. Specifies the number of times a server should retry a *listen* when an attempt to listen on the network fails.

Default value. The default value of this keyword is 3.

Example. To set the number of retries to 5, specify:

```
listenretry=5
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in a multi-user server's communications section (for example, [dbntsrv.ntnbi]) of the configuration file (sql.ini).

locks

Purpose. Specifies the maximum number of lock entries to allocate. SQLBase allocates lock entries dynamically (in groups of 100) on an as-needed basis.

Specify this keyword only when you are trying to protect against a run-away application that is accessing too much data in one transaction.

Default value. The default value of this keyword is 0, which means that there is no limit on the number of locks allocated; as many lock entries can be allocated as memory permits.

Example. To limit lock entry allocation to 500, specify:

```
locks=500
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

locktimeout

Purpose. Specifies how long to wait when acquiring a lock. If an application cannot acquire a lock within the specified period of time, SQLBase returns a timeout error.

Default value. The default value of this keyword is 300 seconds for all servers but SQLBase single-user for Windows. The default for SQLBase single-user for Windows is 2 seconds.

Example. To set the timeout period to 2 minutes, specify:

```
locktimeout=120
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

log

Purpose. Writes all the messages that appear on the server's Process Activity display to a specified file.

This keyword is equivalent to using SQLTalk's SET ACTIVITYLOG command.

These messages are helpful when debugging, but they are not meant to be logged in a production environment. Logging incurs overhead, and the log file can become large very quickly if your site is an active one.

Default value. This keyword does not have a default value.

Example. To write the messages to a file called *debug.txt*, specify:

```
log=d:\test\debug.txt
```

The maximum length of the *log* value is 260 characters. The maximum number of characters allowed in an *log* directory path (excluding the filename which has a limit of eight characters and a three-character extension) is 246. This includes the drive letter, colon, leading backslash, and terminating null character.

Semicolons and commas cannot be part of the pathname. A directory name enclosed in single quotes can contain an embedded single quote which is indicated by TWO single quotes (for example: D:\'''test'''\).

Note: Characters reserved in the Windows environment cannot be used in directory or file names; for example: <> : " / \ |. Words reserved in the Windows environment cannot be used as filenames; for example, *con* or *aux*.

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

logdir

Purpose. Redirects the transaction logs to the specified drive and directory.

When SQLBase creates log files, it appends the database name to the specified path name to ensure that log files for different databases are not placed in the same directory.

Default value. By default, SQLBase creates transaction log files in the home database directory. Redirecting the transaction log files has two benefits:

- If the database and its logs are on separate disk drives, higher transaction throughput is possible because of less disk arm movement and the ability to do parallel disk accesses.
- If the database and log files are on different disk drives, a media failure does not destroy both the logs and the database.

Example. To redirect log files to the d:\test directory, specify:

```
logdir=d:\test
```

The maximum length of the *logdir* value is 260 characters. The maximum number of characters allowed in an *logdir* directory path (excluding the filename which has a limit of eight characters and a three-character extension) is 246. This includes the drive letter, colon, leading backslash, and terminating null character.

Semicolons and commas cannot be part of the pathname. A directory name enclosed in single quotes can contain an embedded single quote which is indicated by TWO single quotes (for example: D:\'"test"'\).

Note: Characters reserved in the Windows environment cannot be used in directory or file names; for example: **<> : " / \ |**. Words reserved in the Windows environment cannot be used as filenames; for example, *con* or *aux*.

Recommended. Configuration of this keyword is recommended.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

logfileprealloc

Purpose. Enables and disables transaction log file preallocation.

Default value. By default, *logfileprealloc* is off (0) and the current log file grows in increments of 10 percent of its current size. This uses space conservatively, but can lead to a fragmented log file which can affect performance.

Setting *logfileprealloc* on (1) means that SQLBase creates log files full size (preallocated) and they do not grow.

Example. To enable log file preallocation, set logfileprealloc=1 in the configuration file (sql.ini):

```
logfileprealloc=1
```

Start SQLTalk and issue a SET LOGFILESIZE command.

From SQLTalk, issue a RELEASE LOG command. This forces SQLBase to start a new transaction log which it creates with the size you specified in the previous step.

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

mainwin

Purpose. Determines the position and size of the main server window.

Example. The syntax is:

```
MAINWIN=status,x,y,cx,cy
```

The following values are valid for these settings:

- ***Status***

  NORM for normal window status, MAX if the window is maximized, and MIN if the window is minimized.

- ***x, y***

  These values represent the position of the window. For Windows NT, this is the window's top-left corner.

- ***cx,cy***

  These values represent the size of the window. For Windows NT, these are the coordinates of the window's lower-right corner. These values are system-dependent; do not manually modify or copy them from one system to another.

Optional. Configuration of this keyword is optional.

Where configured. This keyword is configured automatically in the server's GUI section (for example, [dbntsrv.gui]) when you use the **Save Settings** option in the server's **File** menu. Do not set a value for this keyword manually.

maxnestinglevel

Purpose. Specifies the maximum nesting level for recursing stored procedures.

Default value. The minimum value for this keyword is 1; maximum value is 16. Maximum value is dependent on the limit of memory.

Example. The syntax is:

```
maxnestinglevel=1
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

names

Purpose. Sets the number of installable NetBIOS names.

Default value. This keyword has no default value.

Example. To allocate network name resources for a database server and 5 databases, specify:

```
names=6
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in a Windows NT client NetBIOS communications section or a Windows NT multi-user server IBM NetBIOS communications [dbntsrv.ntnbi]) section, or the Windows 95 client NETBIOS communications section of the configuration file (sql.ini).

Read the section *How SQLBase uses NetBIOS resources* on page *2-6* for detailed information about NetBIOS names.

ndsloginid

Purpose. Specifies the userid required to log into the NDS Tree for the purposes of adding, deleting, or modifying the SQLBase server and database object entries.

Default value. By default, this keyword uses the login context of the NDS user.

Example. To set the NDS login ID to **admin**, specify:

```
ndsloginid=admin
```

Optional. Configuration of this keyword is mandatory only if NDS is used for advertising SQLBase servers and databases. Otherwise this keyword is ignored.

Where configured. For the SQLBase Server for NetWare 4.x, configure this keyword in the server section of the configuration file (sql.ini). This keyword is not applicable to the SQLBase Server for NetWare 3.x.

ndsloginpassword

Purpose. Sets a password for the NDS userid required to log into the NDS Tree for the purposes of adding, deleting, or modifying the SQLBase server and database object entries.

Default value. By default, this keyword uses the login context of the NDS user.

Example. To set the NDS login ID's password to **admin**, specify:

```
ndsloginpassword=admin
```

Optional. Configuration of this keyword is mandatory only if NDS is used for advertising SQLBase servers and databases. Otherwise this keyword is ignored.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

netcheck

Purpose. Enables and disables a checksum feature that detects transmission errors between a client and a server. To use this feature, both the client and the server must enable netcheck.

The checksum feature is enabled on the server side when you bring the server up. When a client attempts to connect to a database on the server, it requests the network check. If the server confirms the request, then the checksum feature is enabled on the client side.

It is the network's responsibility to ensure data integrity. However, the checksum feature can offer additional protection. In cases where the network is unable to detect a problem (such as a memory problem in the communication card), the checksum feature can prevent corrupted data from being passed to either the client or the server.

If an error is detected by either the client or the server, the detector tries to transmit the data again. If an error from the server is detected and the process activity level is 2 or more, SQLBase displays a message indicating its intention to re-transmit. If the error re-occurs, SQLBase returns an error code to the client, aborts the transmission, and displays a message on the server's Process Activity screen. The corrupted data is not processed.

Most networks such as Ethernet, Token-Ring, and ARCNET check for errors using CRC-16 or CRC-32 algorithms. However, an error can occur on the path that the transmitted data takes from the network board to other components in the computer or in network gateways. It is these types of errors that this feature detects.

Using this network checksum feature can decrease performance by as much as 5 to 10 percent.

Default value. By default, *netcheck* is off (0).

Example. To enable *netcheck*, specify:

```
netcheck=1
```

Remember to add this entry to the configuration files on both the server and the client.

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in any client router or multi-user server section of the configuration file (sql.ini).

netchecktype

Purpose. Specifies the algorithm SQLBase uses when *netcheck* is enabled. Configure this keyword only when you enable *netcheck*.

Default value. By default, checksum (0) is enabled.

Example. To switch to CRC/16, specify:

```
netchecktype=1
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in any client router or multi-user server section of the configuration file (sql.ini).

netlog

Purpose. Invokes a diagnostic server utility that records database messages to a specified log file. This utility logs all messages that pass between a server and clients communicating over a network using NetBIOS or SPX.

Do not use *netlog* unless instructed to do so by Centura's Technical Support staff. You must start the netlog utility on a new database or one whose state you know to be uncorrupted. Do not start netlog on a database that you know to have a problem. Starting netlog when a database already has a problem will not help, because the purpose of netlog is to show how the problem happened in the first place. Once you have recorded the problem, the technical staff at Centura can play the netlog file back to reproduce the problem.

***Syntax*.** The syntax of the keyword is:

```
netlog=filename[,force]
```

where ‘filename’ is the name of a file to which the messages will be written and ‘force’ is an optional parameter.

You should use the FORCE parameter for multi-user environments. If you do not use FORCE, the order of your transactions may not be the same in the log replay as they were for the original run.

Default value. By default, the netlog utility is off.

Example. To invoke the netlog utility and write to a file called *dbmsgs.log*:

```
netlog=dbmsgs.log
```

In fatal error situations, you may be asked by your technical support staff contact to add the FORCE option:

```
netlog=dbmsgs.log,force
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in any multi-user server section of the configuration file (sql.ini).

numconnectionecb

Purpose. Used to adjust the number of connection Event Control Blocks (ECBs). Connection ECBs are used by SQLBase to receive connection requests from client programs. If many client programs try to connect to SQLBase at the same time and an error message is displayed (for example, “Cannot open database”), increase this parameter.

Example. To instruct SQLBase to increase the connection ECBs to 15, type:

```
numconnectionecb=15
```

Default value. The default value for *numconnectionecb* is 10.

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

numlistenecb

Purpose. Used to adjust the number of listen Event Control Blocks (ECBs). Listen ECBs are used by SQLBase to receive incoming queries and data from client programs. If many client programs try to communicate with SQLBase at the same time and one or more session appear to be stalled after issuing a query or command, increase this parameter.

Example. To instruct SQLBase to increase the listen ECBs to 30, type:

```
numlistenecb=30
```

Default value. The default value for *numlistenecb* is 25.

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

nwadvertisemode

Purpose. Specifies what mode to use when advertising SQLBase servers and databases on the network. The *NWAdvertiseMode* keyword requires a mandatory setting of 2 if NDS is used for advertising SQLBase servers and databases. Otherwise the default for this keyword is 1, SAP only.

If NDS and bindery emulation reside on the system, a setting of 3, allows SAP advertising on the network so that non-NDS servers and clients can locate SQLBase servers and databases. If SAP is turned off, non-NDS servers and clients relying on SAP will not find the SQLBase server and installed databases.

Note that enabling SAP advertising results in additional network traffic.

Example. To instruct SQLBase to use the *nwadvertisemode* keyword to advertise SQLBase servers and databases only on the NetWare Directory Services tree, specify:

```
nwadvertiseMode= 2
```

Default value. The default value for *nwadvertisemode* is 1.

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

optimizefirstfetch

Purpose. Used to set the optimization method for the first fetch. When set, the keyword instructs the optimizer to pick a query execution plan that takes the least amount of time to fetch the first row of the result set.

The valid values for this keyword are 0 and 1. When *optimizefirstfetch* is set to 0, SQLBase optimizes the time it takes to return the entire result set. When *optimizefirstfetch* is set to 1, SQLBase optimizes the time it takes to return the first row.

Example. To instruct SQLBase to use the *optimizefirstfetch* keyword to optimize the time of the first fetch, specify:

```
optimizefirstfetch=1
```

The setting applies for all queries being compiled for the current cursor, unless a query is compiled under a scope of a cursor with a different optimization mode.

Default value. The default value for *optimizefirstfetch* is 0.

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

optimizerlevel

Purpose. Determines the optimizing techniques that SQLBase uses for all clients that connect to a server.

Default value. By default, *optimizerlevel* is 2 which causes SQLBase to use current optimizing techniques.

If you set *optimizerlevel* to 1, SQLBase uses old optimizing techniques. This setting lets you fall back on old optimizing techniques after upgrading to newer versions of SQLBase. If you discover better performance of a query when *optimizerlevel* is set to 1, you should report it to Centura’s Technical Support team. Be sure not to include compilation time in the comparison of settings 1 and 2.

Example. To tell SQLBase to use old optimizing techniques:

```
optimizerlevel=1
```

You can override the *optimizerlevel* setting for a particular cursor with SQLTalk's SET OPTIMIZERLEVEL command or the *sqlset* API function with the SQLPOPL parameter.

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

oracleouterjoin

Purpose. Turns on and off Oracle-style outer join processing.

Oracle's outer join implementation differs from the ANSI and industry standard implementation. To paraphrase the ANSI standard, the correct semantics of an outer join are to display all the rows of one table that meet the specified constraints on that table, regardless of the constraints on the other table. For example, assume two tables (A and B) with the following rows:

Table A (a int)	Table B (b int)
1	1
2	2
3	3
4	
5	

If you issue the following SQL command:

```
SELECT a, b FROM A, B WHERE A.a = B.b (+) AND B.b IS
   NULL;
```

the ANSI result is:

Table A (a int)	Table B (b int)
1	
2	
3	
4	
5	

Assuming the same two tables and the same SQL command, the correct result for Oracle is:

Table A (a int)	Table B (b int)
4	
5	

Default value. By default, *oracleouterjoin* is off (0) and SQLBase implements outer joins according to the ANSI standard.

Example. To direct SQLBase to process outer joins the Oracle way, specify:

```
oracleouterjoin=1
```

If you set *oracleouterjoin* on (1), you receive the Oracle result shown above.

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

osavgwindow

Purpose. Specifies the number of samples of the CPU % Utilization value to keep for determining the average value. You can specify a window size of 1 to 255;

Default value. By default, *osavgwindow* is 1.

Example. To specify a window size of 25:

```
osavgwindow=1
```

You can override the *ossamplerate* with the *sqlset* API function with the SQLPAWS parameter.

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

ossamplerate

Purpose. Specifies the frequency at which operating system statistics (CPU % Utilization) are gathered. You can specify a setting of 0 to 255 seconds.

Default value. By default, *ossamplerate* is zero (0), which disables the gathering of CPU statistics.

Example. To tell SQLBase to gather statistics every 10 seconds:

```
ossamplerate=1
```

You can override the *ossamplerate* with the *sqlset* API function with the SQLPOSR parameter.

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

outmessage

Purpose. Sets the size (in bytes) of the output message buffer. The output message buffer holds output from the application (such as a SQL command to compile or rows of data to insert into a database).

There is one output message buffer per connected cursor on the client computer. The database server maintains a buffer that is the size of the largest output message buffer on the client computer. The client builds a message in its buffer, then sends it across the network to the server's output message buffer. It is called an output message buffer because it is output from the client's perspective.

Default value. By default, SQLBase maintains an output message buffer large enough to hold any SQL command or row of data. Despite the *outmessage* value, SQLBase dynamically allocates more space if necessary.

A large output message buffer does not necessarily improve performance because the buffer only needs to be large enough to hold the largest SQL command to compile or the largest row of data to insert. (Rows are always sent to the database and inserted individually except in bulk execute mode.) A large output message buffer can allocate space unnecessarily on both the client and the server, and it does not reduce network traffic unless bulk execute is on.

Example. To set the output message buffer to 2000 bytes, specify:

```
outmessage=2000
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in any client router section of the configuration file (sql.ini), except for Windows NT.

partitions

Purpose. Enables access to partitioned databases.

Default value. By default, *partitions* is off (0) which means that:

- You can restore *main.dbs*.
- You cannot access partitioned databases. An exception is when you do not have a MAIN database. After creating the first dbarea, SQLBase creates *main.dbs* and sets partitions on (1).

Example. To enable access to partitioned databases, specify:

```
partitions=1
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

password

Purpose. Sets a password for the server.

Specify the *password* keyword immediately after the *servername* keyword to force users to specify a password to perform administrative operations or to get server information.

Default value. The default password is SYSADM. A server password can be no longer than eight characters.

Example. To set server1’s password to ‘secret’, specify:

```
servername=server1
password=secret
```

Optional. Configuration of this keyword is optional, but recommended for production databases.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

patch

Purpose. Use this keyword and value when running SQLBase Server for NetWare on a NetFrame machine.

Default value. This keyword has no default value.

Example. To enable this keyword, specify:

```
patch=1
```

Mandatory/Unnecessary. Configuration of this keyword is mandatory if you are running SQLBase on a NetFrame computer; otherwise it is unnecessary.

Where configured. Configure this keyword in the [patch] section of a NetFrame computer's NetWare database server configuration file (sql.ini).

preferrednameservice

Purpose. Specifies a name service to use. A name service is something that converts a database name into an address and socket combination in order to establish a session with a database server. Setting this keyword reduces the time necessary for making a connection.

Specify the name service (Directory Services or Bindery) you prefer to use:

- Directory Services

  Find the database using directory services first, then look for it in the bindery.

- Bindery

  Disables directory services (as well as the corresponding logins that it would do).

Default value. For the NetWare 4.0 environment, the default is directory services, followed by bindery. For the NetWare 3.11 environment, the default is BIN if no keyword is specified in the client section.

The syntax is:

```
preferrednameservice=[NDS|BIN]
```

If you do not set this keyword, SQLBase attempts to make a decision based on the following:

- Is directory services running?

  If yes, use it to find the database.

- If no, try the NetWare bindery. Was the name found?

  If yes, make the connection.

Example. Here is an example of entries for the *searchcontext* and *preferred nameservice* keywords:

```
[winclient.spxw4]
searchcontext=CN=ADMIN.O=Centura
preferrednameservice=NDS
```

The *preferrednameservice* keyword also applies to the NetWare 3.11 platform.

Since directory services is not a valid option for the NetWare 3.11 platform, your only option for configuring the *preferrednameservice* keyword in the [winclient.spx] section is 'bin'.

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in any of the following client sections of the sql.ini configuration file:

- SPX Windows client sections: [winclient.spxw4], [win32client.spx32], and [win32client.wsspx]
- TCP/IP Windows client sections:[nwclient.tip], [winclient.wsock], and [win32client.ws32]

procwin

Purpose. Determines the status of the Process Activity subwindow. The syntax for this window keyword is identical to that for the ***mainwin*** keyword, with the addition of a ***closed*** status. This indicates that the given subwindow was closed when you last used the **Save** option of the **File** menu.

Example. The format of these keywords is:

```
procwin=status,x,y,cx,cy
```

Optional. Configuration of this keyword is optional.

Where configured. This keyword is configured automatically in the server’s GUI section (for example, [dbntsrv.gui]) when you use the **Save Settings** option in the server’s **File** menu. Do not set a value for this keyword manually.

readonly

Purpose. Allows users connecting to *any* of the databases on the server to use the read-only isolation level.

To enable read-only access to a *specific* database on the server, use SQLTalk's SET READONLY command or the *sqlset* API function with the SQLPISO parameter.

The read-only isolation level allows for a consistent view of data; for the duration of your session, the data in the database appears not to change. When readonly is enabled, SQLBase maintains a read-only history file that contains multiple copies of modified database pages. When you try to access a page which has been changed by another user, SQLBase retrieves a copy of the original page from the history file.

Default value. By default, *readonly* is disabled (0) and users are prevented from going into read-only mode because read-only transactions can affect performance.

Example. To enable read-only access, specify:

```
readonly=1
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

retry

Purpose. Specifies the number of times a Windows NetBIOS client or a Windows SPX client should attempt to connect to a server. Its purpose is to allow for the brief period that the server cannot respond to the next client's connection request because it is responding to a current connection request.

Default value. By default, the retry count is 3. You should only increase the value of this keyword if you get error messages when trying to connect to a database.

The number of retries necessary varies for each network and can only be determined by observation.

Example. To set *retry* to 6, specify:

```
retry=6
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in any of the following sections of the configuration file (sql.ini):

- Windows client: [winclient.nbiow], and [winclient.spxw4]
- NetWare database server: [<*servername*>.spx]. For example: [dbnwsrv.spx]

retrytimeout

Purpose. Specifies how long a Windows client using either the NetBIOS or SPX protocol should wait for a response from a server when attempting to connect.

Default value. By default, *retrytimeout* is 10. The meaning of this setting depends on the speed of the client computer.

Setting *retrytimeout* to zero (0) means to try the connection only once (no matter how many seconds it takes).

Example. To set the *retrytimeout* value to 20, specify:

```
retrytimeout=20
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in a Windows client's [winclient.nbiow]or [winclient.spxw4] section of the configuration file (sql.ini).

searchcontext

Purpose. Specifies what part of the NDS Tree that an NDS client requires to search for the SQLBase server name and installed database names. You can specify more than one searchcontext. Separate each searchcontext by a period.

If two or more keywords are specified, the search begins from the first to the last searchcontext, and if the object is not found, the current context of the logged in user is searched. If the client is non-NDS, the keyword is ignored.

Note: Be sure the *searchcontext* keyword is set to the same value as the *insertioncontext* keyword in the [server] section of sql.ini.

Default value. If no keyword is specified in the client section, the entire NDS Directory Tree is searched.

Example. To specify that Windows NT SPX clients search a part of the Directory Tree identified as O=CENTURA, specify:

```
[winclient.spxw4]
searchcontext=O=CENTURA.
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in any of the following client sections of the sql.ini configuration file:

- SPX Windows client sections: [winclient.spxw4], [win32client.spx32], and [win32client.wsspx]
- TCP/IP Windows client sections:[nwclient.tip], [winclient.wsock], and [win32client.ws32]

servername

Purpose. Specifies a name for a multi-user server.

To perform administrative operations, a user must establish a connection to the server itself and specify the server's password (if one exists). This prevents unauthorized users from performing destructive operations.

To establish (or break) a connection to a server, use SQLTalk's 'SET SERVER <servername>' command.

SQLTalk's SHOW DATABASES command also accepts a server name.

The *sqlcsv* API function requires a server name as input and returns a handle which the *sqldsv* functions requires as input. The *sqldbn* API function requires a server name as input and returns a list of databases on a server.

Default value. If you don’t configure the *servername* keyword, it defaults to s### where ‘###’ is a random 3-digit number.

The name can be no longer than eight alpha-numeric characters, it must begin with a letter, and it must be unique on the network.

Example. To name a server ‘server1’, specify:

servername=server1

You can optionally specify one or more communications library names after the server name. For example:

servername=server1,sqlntnbi

You must also specify the communications libraries with the *comdll* keyword in the server's [*.dll] section.

Recommended. Configuration of this keyword is recommended for those sites for which security is important.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

servernames

Purpose. Specifies the names of one or more LAN Manager file servers to which a database server connects.

Default value. This keyword has no default value.

Example. To identify two file servers, specify:

servername=lanman1;lanman2

Substitute appropriate LAN Manager server names for *lanman1* and *lanman2*.

Mandatory/Unnecessary. Configuration of this keyword is mandatory in a LAN Manger environment for remote connections; otherwise, it is unnecessary.

Where configured. Configure this keyword in the Named Pipes section of a SQLBase for Windows NT server’s configuration file ([dbntsrv.npipe]) or the [win32client.npipe] section of a Windows NT client’s configuration file (sql.ini).

serverpath

Purpose. Tells a client using the TCP/IP protocol how to locate SQLBase servers by associating a server name and one or more database names with a host's network name, IP address, or TCP port number. You also must specify at least one serverpath keyword in a server section if the server is connected via TCP/IP and is participating in a distributed transaction.

The format of the keyword is:

```
serverpath=servername,hostname[,tcp-port]/
dbname[,dbname...][*]
```

The servername is the name of the server as specified in the server's sql.ini file.

The hostname is the network name or the IP address of the machine on which the server is running.

The optional tcp-port is the TCP port number on which the host is listening for connection requests. You need to specify this only if the *listenport* keyword is specified in the TLI-TCP/IP section of the server's sql.ini file.

The dbname is the name of a database on the server. You can specify more than one database name by separating each name with a comma. The asterisk (*) is an optional wildcard symbol that you can use to match any database name.

Examples. Here are two examples of serverpath statements:

This example shows that a SQLBase server called *server0* is installed on a host called *host1* and is listening on the databases *demo* and *demox* for connection requests.

```
serverpath=server0,host1/demo,demox
```

This example shows that a SQLBase server called *server1* is installed on a host with an IP address of 151.222.1.1, and is listening on the database account for connection requests.

```
serverpath=server1,151.222.1.1/account
```

The following example shows the use of multiple *serverpath* statements. In this example, if the client attempts to connect to database *ABC*, SQLBase matches ABC to the wildcard (*) and then uses the second *serverpath* (and its IP address) to connect to it. However, if the client attempts to connect to *DEMO1*, the first *serverpath* is used.

```
serverpath=server1,198.206.11.10/demo1

serverpath=server2,198.206.11.20/demo2,*
```

Mandatory/Unnecessary. Configuring this keyword is mandatory for clients accessing SQLBase servers through the TCP/IP protocol, or for servers participating in a distributed transaction through TCP/IP. Otherwise, it is unnecessary.

Where configured. Configure this keyword in a sql.ini client section that supports TCP/IP communications. Currently, this includes the following sections:

- [WINCLIENT.WSOCK] for 16-bit Windows Sockets TCP/IP clients.
- [WIN32CLIENT.WS32] for 32-bit Windows Sockets TCP/IP clients.
- [NWCLIENT.TIP] for NetWare TLI-TCP/IP clients or servers (communicating via TLI-TCP/IP) participating in a distributed transaction.
- [DBNTSRV.WS32] for 32-bit Windows servers participating in a distributed transaction using TCP/IP under Windows Sockets.

You can configure multiple *serverpath* keywords in the client's Windows Socket section of sql.ini. This gives the client the flexibility to connect to databases on different servers from the same session. However, the following restrictions apply:

- Each *serverpath* must specify a unique host name or IP address.
- The database names specified in the *serverpath* statements must be unique in that section. For example, you cannot specify *demo* as a database name on two *serverpath* statements in the client's Windows Socket section.
- You cannot use more than one wildcard character (*) in the section. That is, you cannot specify two *serverpath* statements, each with the wildcard (*) in it.

serverprefix

Purpose and default value. Internally, SQLBase needs to distinguish server names from database names. To do this, SQLBase adds a space to the beginning of server names. However, this space can cause a problem for network routers, bridges, and gateways. The *serverprefix* keyword lets you change the character that SQLBase uses as the prefix for server names.

Example. Specify one character for the *serverprefix* value, for example:

```
serverprefix=$
```

If you have a *serverprefix* entry in a server's configuration file, all clients connecting to databases on that server must specify the same prefix in their configuration files.

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in a client router NetBIOS communications section or a server's NetBIOS communications section of the configuration file (sql.ini).

sessions

Purpose. Sets the maximum number of sessions that can be allocated for a process.

A NetBIOS session is a connection between two applications which exchange a series of messages in order to complete a task. A session can exist for extended periods. Such a connection is sometimes called a virtual circuit. A multi-user database server can have multiple simultaneous sessions servicing many clients.

This is how the server program uses NetBIOS sessions:

- The server program itself uses one session.
- Each database that the server listens on uses one session.
- Each connected client application uses one session.

Each connected client application, regardless of the number of connected cursors, uses one session. The server uses a temporary session when an application connects a new cursor, but once the connection is established, it frees the temporary session.

Default value. This keyword has no default value.

Optional. Configuration of this keyword is optional.

Note: The *sessions*, *commands*, and *names* keyword values cannot exceed the values defined in the IBM configuration file. Reconfiguring these keywords can require you to adjust both the IBM network configuration file and Centura's configuration file.

Where configured. Configure this keyword in a Windows NT multi-user server's IBM NetBIOS communications ([dbntsrv.ntnbi]) section of the configuration file (sql.ini).

showmaindbname

Purpose. Enables (1) or disables (0) the display of the MAIN database when you query the server for a list of database names.

Default value. By default, the display of the MAIN database is off (0).

Example. To enable the display of the MAIN database, specify:

```
showmaindbname=1
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in any client router section of the configuration file (sql.ini).

silent

Purpose. Turns the display for a multi-user server on and off.

Default value. By default, multi-user server displays are on (0).

Example. To set the display of the server screens off, specify:

```
silent=1
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in any server section of the configuration file (sql.ini).

sortcache

Purpose. Specifies the number of 1K (1024-byte) pages of memory to use for sorting. The value applies across the server. Sorting is done when you specify a DISTINCT, ORDER BY, GROUP BY, or CREATE INDEX clause, or when SQLBase creates a temporary table for join purposes.

Note that the combined values of *cache* and *sortcache* cannot exceed the memory resources of your system.

Default value. The default is:

- For *dbwservr.exe*: 1000
- For all other platforms: 2000

(SQLBase accepts a maximum value of one million, but you should set this value based on how much memory you have in your system).

Example. To set the amount of memory to use for sorting to 1000, specify:

```
sortcache=1000
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

Note: Sortcache memory is not pre-allocated. The memory used is separate from the database cache, so ensure there is enough server memory for both.

statwin

Purpose. Determines the status of the Server Status subwindow.

The syntax for this keyword is identical to that for the ***mainwin*** keyword, with the addition of a ***closed*** status. This indicates that the given subwindow was closed when you last used the **Save** option of the **File** menu.

Example. The format of this keyword is:

```
statwin=status,x,y,cx,cy
```

Optional. Configuration of this keyword is optional.

Where configured. This keyword is configured automatically in the server's GUI section (for example, [dbntsrv.gui]) when you use the **Save Settings** option in the server's **File** menu. Do not set a value for this keyword manually.

syswin

Purpose. Determines the status of the System Activity subwindow.

The syntax for this keyword is identical to that for the ***mainwin*** keyword, with the addition of a ***closed*** status. This indicates that the given subwindow was closed when you last used the **Save** option of the **File** menu.

Example. The format of this keyword is:

```
syswin=status,x,y,cx,cy
```

Optional. Configuration of this keyword is optional.

Where configured. This keyword is configured automatically in the server's GUI section (for example, [dbntsrv.gui]) when you use the **Save Settings** option in the server's **File** menu. Do not set a value for this keyword manually.

tempdir

Purpose. Specifies the directory where SQLBase places temporary files.

In the course of processing, SQLBase can create several kinds of temporary files: sort files, read-only history files, and general-use files.

Temporary files have names in the format: *sqlxxxx.tmp* where 'xxxx' is a randomly-generated number.

SQLBase uses asynchronous input/output to read and write temporary files.

Default value. SQLBase places temporary files in one of three places in this order:

1. The directory specified by the configuration file's *tempdir* keyword.

2. The directory specified by the *tempdir* environment variable.

3. The current directory.

Example. To specify d:\tmp as the temporary directory, specify:

```
tempdir=d:\tmp
```

The maximum length of the *tempdir* value is 260 characters. The maximum number of characters allowed in an *tempdir* directory path (excluding the filename which has a limit of eight characters and a three-character extension) is 246. This includes the drive letter, colon, leading backslash, and terminating null character.

Semicolons and commas cannot be part of the pathname. A directory name enclosed in single quotes can contain an embedded single quote which is indicated by TWO single quotes (for example: D:\'"test"'\).

Note: Characters reserved in the Windows environment cannot be used in directory or file names; for example: <> **:** **“** / \ |. Words reserved in the Windows environment cannot be used as filenames; for example, *con* or *aux*.

Mandatory/Optional. You must set *tempdir* for read-only databases. Otherwise, configuration of this keyword is optional, though recommended for production sites to ensure that you do not accidently run out of disk space.

Where configured. Configure this keyword in any multi-user server section of the configuration file (sql.ini).

threadmode

Purpose. Specifies whether to use native threads or SQLBase threads for NetWare or Windows NT. For Windows 95, SQLBase now uses Windows 95 native threads only.

A value of 1 indicates SQLBase threads and a value of 2 indicates native threads for the current platform.

Default value. By default, *threadmode* is 1, except on Windows 95 where the default is 2.

On NetWare platforms, if you are running in Ring 0, Centura recommends using SQLBase threads which invoke stack switching. This should yield better performance. Novell disallows stack switching in Ring 3, so be sure to set *threadmode* to 2 when in Ring 3.

Example. To use native threads, specify:

```
threadmode=2
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the [dbnw*] section of a SQLBase for NetWare 4.x, or [dbntsrv] section of a SQLBase for Windows NT server's configuration file (sql.ini).

threadstacksize

Purpose. Specifies the stack size (in bytes) on NetWare 4.x.

Default value. By default, *threadstacksize* is 10 kilobytes and the minimum value is 8192 bytes.

You should not decrease the default value. Running complex queries when *threadstacksize* is set to 8192 can result in a stack overflow error.

If you receive stack overflow errors, increase the value of *threadstacksize* by 512 bytes at a time.

Example. To increase the stack size to 8704 bytes, specify:

```
threadstacksize=8704
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the [dbnw*] section of a SQLBase for NetWare 4.x server's configuration file (sql.ini).

timecolononly

Purpose. Configures SQLBase to recognize when a delimiter other than a colon (:) is being used to separate the hours, minutes, and seconds portions of a time value.

For example, when *timecolononly* is on (1), a colon is the only delimiter SQLBase recognizes. If SQLBase encounters any other character being used as the delimiter in the time value, it produces an error.

If you want to enter a date value as dd.mm.yyyy, set *timecolononly* on (1) so that SQLBase will not interpret the value as a time value (which it would do, since the year portion of the value is less than three characters).

Default value. The default is off (0).

Example. To enable SQLBase to recognize a delimiter other than a colon, specify:

```
timecolononly=0
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

timeout

Purpose. Specifies the time period (in minutes) that the server waits for a client to make a request. If the client does not make a request within the specified period, SQLBase rolls back the client session, processes, and transactions. The timeout clock restarts each time the client makes a request.

This keyword is transaction-specific, not cursor-specific.

Default value. The *timeout* value is 0 (infinite) by default, and the maximum value is 200 minutes.

Example. To set a timeout period of one hour, specify:

```
timeout=60
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

timestamps

Purpose. Specifies the timestamp setting. Timestamps can be 0 (off) or 1 (on), indicating whether timestamps are displayed with each line of information output to the Process Activity window. Set a timestamp with either the Level menu on the GUI server, the SQLTalk SET TIMESTAMPS command, or the ***sqlset*** SQL/API call in conjunction with the SQLPTMS parameter.

Optional. Configuration of this keyword is optional.

timezone

Purpose. Sets the value of SYSTIMEZONE, a SQLBase keyword that returns the time zone as an interval of Greenwich Mean Time. SYSTIMEZONE uses the expression (SYSTIME - TIMEZONE) to return the current time in Greenwich Mean Time.

Calculate the value of *timezone* as an interval in hours, where the interval is local time minus Greenwich Mean Time.

Default value. By default, *timezone* is 0.

Example. When it is 1:00pm in the Pacific Standard Timezone (PST), it is 9:00pm in Greenwich. This equates to 1 minus 9 which equals -8. Set:

```
timezone=-8
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

users

Purpose. Specifies the maximum number of client applications (processes) that can connect to a server simultaneously. This means, for example, that a server configured with *users=5* could support five clients running one application each, or one client running five applications, or two clients — one running two applications and the other running three applications, and so on.

Default value. The default value of *users* is 128, and the maximum is 800.

Example. To limit server access to 50 applications, specify:

```
users=50
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in any multi-user server section of the configuration file (sql.ini).

workalloc

Purpose and example. Specifies the basic allocation unit of a work space. For example, if a SQL command requires 5000 bytes and the default value of 1000 is in effect, SQLBase makes 5 memory allocation requests to the operating system (5 * 1000 = 5000).

If, on the other hand, you specify:

```
workalloc=2500
```

SQLBase makes only 2 memory allocation requests to the operating system (2 * 2500 = 5000).

Default value. The default is 1000 bytes.

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

worklimit

Purpose. Specifies a maximum memory limitation (in bytes) for SQL commands. SQLBase cannot execute any SQL command requiring more than 4000 bytes of memory.

Default value. The default is NULL, meaning that no memory limitation exists.

Example. To set the maximum memory limitation, specify:

```
worklimit=4000
```

Optional. Configuration of this keyword is optional.

Where configured. Configure this keyword in the server section of the configuration file (sql.ini).

Chapter 4

Databases

This chapter discusses various aspects of databases that a DBA needs to know:

- About databases
- Database cache
- Database files (transaction log files and temporary files)
- Partitioned databases
- Database auditing

About databases

This section describes the structure, naming, and placement of databases in SQLBase.

Database template (start.dbs)

SQLBase has a database template file called *start.dbs*. Each new database that you create is a copy of *start.dbs.*

This template file contains the system catalog tables that SQLBase maintains. It also reflects any changes to the database structure; an example is the changes to the structure between one version of SQLBase to the next.

If you are doing either of the following upgrades, you need to unload your old database files and then load them to the new SQLBase server:

- From a version prior to 6.0.
- From one platform to another (such as Windows to Windows NT).

Database names

SQLBase database names must conform to the following restrictions:

- They can be up to eight characters in length.
- They must begin with a letter and include only letters and numeric digits.
- They must have a *.dbs* file extension.

Unless you are manually copying the *start.dbs* file to a new file, do not specify a file extension; SQLBase automatically assigns each database a file extension of *.dbs*. For example, if you execute:

```
CREATE DATABASE test;
```

from a SQLTalk session, SQLBase creates a database called *test.dbs* in a subdirectory called \TEST. If the value of the *dbdir* keyword is C:\CENTURA, SQLBase places *test.dbs* in the C:\CENTURA\TEST subdirectory.

dbname keyword

For all servers except Windows, you must configure a *dbname* keyword for each database on the server. Conversely, make sure that each *dbname* keyword in the server's configuration file (sql.ini) has a corresponding database in the appropriate directory. If there is a *dbname* keyword whose associated database file does not exist, you receive error 401 (‘Cannot open the database’) when you try to connect to the database.

Issuing a CREATE DATABASE command automatically enters the database name in the sql.ini file.

Default database name, user name, and password

You must specify a database name, user name, and password when logging on to SQLBase. Client programs from Centura like SQLTalk prompt you for these values when you attempt to connect to a database.

SQLBase determines the database name, user name, and password by checking the following entries in this order:

1. What you enter when prompted by a client program.
2. The *defaultdatabase*, *defaultuser*, and *defaultpassword* keywords in sql.ini.
3. The default of DEMO, SYSADM, SYSADM.

Use the Centura Configuration Administrator if you are on Windows NT or Windows 95, or your preferred text editor to change the *defaultdatabase*, *defaultuser*, and *defaultpassword* keywords in the section of sql.ini for the local database server or router.

Database size

SQLBase inposes a 2 gigabyte size limit for non-partitioned databases. If you try to extend a non-partitioned database beyond 2 gigabytes, SQLBase issues an error message. This also means that if you have a partitioned database greater than 2 gigabytes, you cannot create its backup to a single backup file because that file would then exceed 2 gigabytes.

Note: The 2 gigabyte size limit is imposed on all other files manipulated by SQLBase. For example, you cannot create a partition (dbarea) greater than 2 gigabytes, or back up a partitioned database greater than 2 gigabytes to a single backup file.

To support backups of databases greater than 2 gigabytes, SQLBase lets you perform backups to multiple segments. Read *Chapter 8, Backing Up and Restoring Databases* for more information.

SIZE parameter

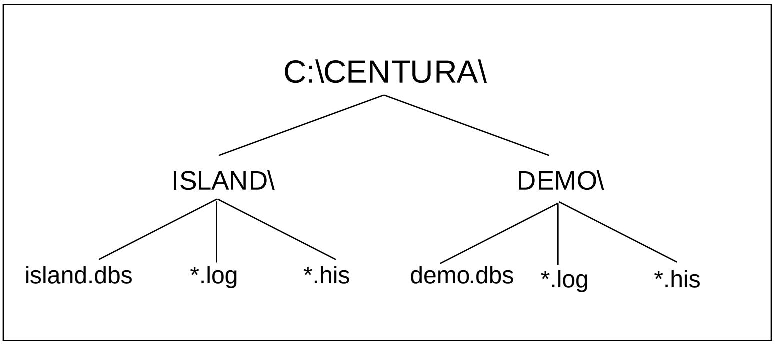
SQLBase uses the SIZE parameter for specifying the size of databases or files in specific operations. Depending on the operation, size can be specified in kilobytes or megabytes. For further details, read the following topics in the *SQL Language Reference* manual:

- CREATE DBAREA
- ALTER DBAREA

- Control filename
- Audit filename

Database location

SQLBase requires the existence of at least one ‘home’ database directory. By default, this is C:\Centura. For each non-partitioned database you create on the server, SQLBase creates a subdirectory within the home database directory. The name of the subdirectory is the same as the database (minus the file extension). The subdirectory contains a database (*.dbs*) file, transaction log files (*.*log*), and may contain read-only history files (*.*his*) for a database.



Default database directory organization

The dbdir environment variable and keyword

You can place databases on different drives and in different home database directories by configuring either the *dbdir* environment variable or the *dbdir* configuration keyword.

In addition to editing the configuration file (sql.ini) directly, both SQLTalk and the SQL/API let you change the value of the *dbdir* keyword. Use SQLTalk’s SET DBDIR command or the SQL/API’s *sqlset* function with the SQLPDBD parameter to do so.

You can specify a path with one or more directories. The maximum length of the *dbdir* value is 255 characters. The format is the same as for the PATH command:

```
dbdir=d:\centura;d:\devel;d:\test
```

For SQLBase Server for NetWare, be sure to include the volume (for example, db:):

```
dbdir=db:\centura
```

You should specify a home database directory with a drive letter and path name. If you omit the drive letter, the current drive is assumed.

SQLBase looks for a database in these places in this order:

1. The directory pointed to by the *dbdir* configuration keyword or by the *dbdir* environment variable.
2. The current directory.
3. The \SQLBASE directory on the current drive.
4. The root directory on the current drive.

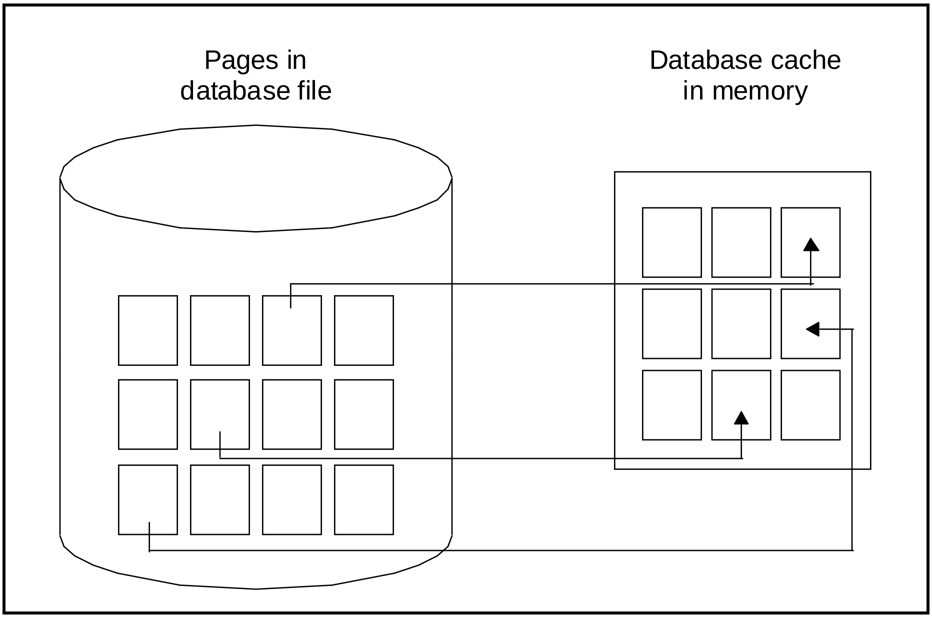
SQLBase creates a new database in the first directory on the *dbdir* path or in the current directory if *dbdir* is not specified.

If the configuration file contains two *dbdir* keywords, the second setting overrides the first.

For partitioned databases, the *dbdir* keyword specifies the directory that contains the MAIN database. In turn, the MAIN database holds the information about where partitioned databases and their log files are stored.

Database cache

SQLBase uses a cache to optimize database input and output. The cache is an area of computer memory on the database server machine that contains copies of pages that users are reading or writing.



Database pages in cache

Pages

A page is a unit of data from the database file that contains one or more rows of table or index data. It is the basic unit of physical disk storage in a database. Pages are stored as linked lists.

A page is 1024 bytes (1 Kbyte) in size.

When a user reads or writes a row or index, SQLBase checks to see if the page in which it is located is already in cache. If the page is not in cache, SQLBase copies it to the cache. If the page is already in cache, SQLBase uses the copy in cache. This reduces disk input and output. It is much faster to read or write to computer memory than to read or write to a disk file.

Committing

When an implicit or explicit commit happens, SQLBase writes a commit record to the transaction log on disk. However, the database page in cache is written back (flushed) to the *.dbs* file based on an LRU (least-recently used) algorithm. The information in the transaction log is enough to completely recreate the update to the database if there is a crash, so there is no need to flush the page from cache immediately after a commit.

Checkpoint time interval

A fuzzy checkpointing scheme minimizes the time to perform crash recovery. *Fuzzy checkpointing* means that modified database cache pages are marked at one checkpoint and then written at the next if they have not already been written by normal cache management. A transaction log record is written which describes the active transactions at each checkpoint. Logs containing the last two checkpoint log records are pinned to the disk (not deletable), as these checkpoints are used to quickly establish a redo start point for the rollforward phase of crash recovery.

The checkpoint time interval parameter governs how often a recovery checkpoint operation is done. You can set the checkpoint time interval with SQLTalk's SET CHECKPOINT command or the SQL/API's *sqlset* function. The default is every minute.

Depending on the applications running against the server, a checkpoint operation may affect performance. In this case, you can increase the checkpoint interval until the desired performance is gained.

Database files

In addition to an open database file, SQLBase may need to open the following files as well:

- Transaction log files
- Temporary files
- Error file

Remote servers may also have these files open:

- Process Activity log file
- Netlog file

Transaction log files

A transaction log file contains before- and after-images of changes to the database as well as log records for transaction control (checkpoints, for example). Transaction control includes documenting when a transaction started and how it ended. COMMIT and ROLLBACK are examples of the latter. A transaction log has three functions:

- Rollback

  Transaction log files contain the data needed to rollback a transaction. A rollback can be initiated by either SQLBase or the user.

- Crash recovery

  Transaction log files contain data necessary to bring a database back to a consistent state after a crash which can occur as the result of a power failure or operator error. A database is in an inconsistent or crashed state if it was not shut down gracefully; an example is when you reboot or power off the server computer without first exiting SQLBase using the Esc key after all sessions have exited. SQLBase performs crash recovery automatically whenever a user connects to a crashed database that has just been brought back online.

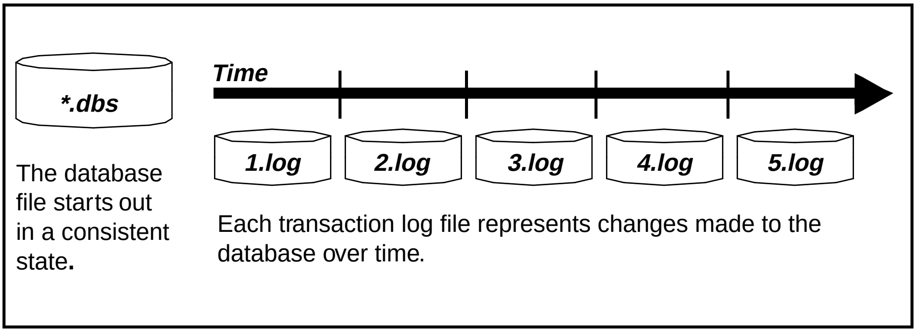
- Media recovery

  Transaction logs, when saved in backup form along with a database, contain data needed to restore that database if damaged by a media failure. A media failure can occur as the result of a crashed hard drive, or a number of other hardware problems. In a situation like this, you can only achieve full recovery through the static restore of an operating system backup.

Transaction logs help ensure data consistency. If a transaction is rolled back or if a system or media failure occurs, SQLBase uses the transaction logs to restore the database to its original state.

For the reasons listed previously, transaction log files, like database files, should be backed up regularly so that they can be restored if necessary.

Note: Never delete transaction log files. SQLBase automatically deletes log files either when they are backed up, or when they are no longer needed for transaction rollback or crash recovery, depending on whether LOGBACKUP is on or off. A database (.dbs) file is useless without its associated log files.



The database file starts out in a consistent state.

Each transaction log file represents changes made to the database over time.

SQLBase creates a new log file upon the first connect to a database. As changes occur to the database, SQLBase writes log records documenting those changes in the active log file.

SQLBase creates a new log file when the current log file reaches its specified maximum size (see *Log file size* on the next page).

The first log file is named *1.log*. Subsequent log files are named sequentially, so *2.log* is created after *1.log*, *3.log* is created next, and so on. The highest log file number is *99999999*. Internally, log files contain timestamps and other values used to ensure that SQLBase can identify them in the correct order. Nevertheless, a validity check is done to verify that the log file numbers are correct, so log files should never be renamed or renumbered.

SQLBase uses the Write-Ahead Log (WAL) protocol that ensures that the log records for a modified page are written before the modified page (in database cache) is written back to the disk. This ensures that SQLBase can undo any uncommitted changes that have been written to disk if there is a crash.

Log file directory

SQLBase creates log files for non-partitioned databases in the home database directory by default. To improve performance, you can redirect the log files to a different directory using the *logdir* keyword in sql.ini. Within this directory, subdirectories are created for each database, and it is in these subdirectories that the database log files are placed by SQLBase.

Partitioned databases do not use the *logdir* keyword. The MAIN database has information about where a partitioned database's log files are stored.

If you redirect log files to another drive, that drive should be on the database machine and should not be on a network file server. This recommendation is based on the reasoning that you will still gain fault tolerance, an increased capacity for log files, and a reduced chance of losing log files if the drive that the database is on fails. The advantage of redirecting log files to a database machine drive is that performance will not suffer the overhead incurred by increased network traffic that results when log files are redirected to a network file server drive.

Log file size

The maximum log file size is set with the SQLTalk SET LOGFILESIZE command or the *sqlset* API function. When the log file grows past this size, SQLBase closes it and creates a new log file. The default size of the log file is 1 megabyte. The smallest size is 100,000 bytes.

A large log file will improve database performance slightly because log files will need to be created less often, but if the log file is too large, it wastes disk space.

Log file growth (incremental or preallocated)

By default, the current log file grows in increments of 10 percent of its current size. This uses space conservatively but can lead to a fragmented log file which may adversely affect performance.

By enabling log file *preallocation*, log files are created full size and do not grow incrementally. You enable preallocation with the *logfileprealloc* keyword in sql.ini, or with the SQLTalk SET LOGFILEPREALLOC command, or the API function *sqlset*.

Log file space availability

For the Windows 32-bit environments and NetWare 3.x or higher, SQLBase queries the disk space available *before* opening a new log file when the current log file for a non-partitioned database has reached its specified maximum size. If the space available is inadequate to ensure database integrity should a system failure occur, SQLBase selectively rolls back any transaction that is causing excessive logging. Transactions that are pinning the earliest active log file are rolled back and the following error message is displayed:

```
03926 LOG LSC Log disk space critically short
```

When this message is displayed, check the log disk space immediately. SQLBase must unpin the log files to free up disk space and to do this, must roll back the transaction that is currently pinning the log files.

When SQLBase rolls back the transaction, it requires free disk space to process:

- Information from logs files the transaction has pinned that must be released to preserve the integrity of the database.
- Generation of new log files to reflect the rollback.

If unpinning the log files associated with the transaction fails to recover log disk space, SQLBase is forced to shut down.

Note: Although you may find you still have disk space, it may not be adequate to fulfill SQLBase recovery requirements. The space required for recovery is twice the size of the data required for processing. SQLBase also requires space allowance for a second recovery attempt, should the first fail.

If you experience unwarranted transaction rollbacks, you can disable SQLBase from checking for available log file space. For details, read the description for the *disablelogspacecheck* keyword in *Chapter 3, Configuration (sql.ini)*.

SQLBase checking for available disk space does not apply to partitioned databases. This is because partitioned databases use “private” logging disk space. In a partitioned database, you can extend the logging space which can span multiple drives.

Transaction span limit

Long running transactions pin log files to disk that could otherwise be released (and deleted if LOGBACKUP is off). You can limit the number of logs pinned down by active transactions by specifying the transaction span limit with the SQLTalk SET SPANLIMIT command or the API function *sqlset*. If you set a limit, SQLBase will roll back long running transactions that exceed the limit, thus allowing pinned log files to be released.

Note: It is important that you understand the transactions at your site before enabling this feature.

You specify the transaction span limit as the number of log files that an active transaction can span. As new log files are created, SQLBase checks this limit for all active transactions. Any transaction that violates the limit is rolled back.

By default, transaction span limit checking is disabled.

Pinned log files

Log files are pinned (held on disk) for several reasons:

- The log file is currently active.
- The log file(s) is needed to potentially rollback one or more active transactions.

  For example, if the earliest active transaction was first logged in log 11 and processing is currently at log 16, logs 11 through 16 are pinned.
- The log file is needed for crash recovery if a failure occurs.

  This is partly covered by the preceding requirement, since crash recovery would need to rollback any transactions active at the time of the crash. In addition, logs containing the last two checkpoint log records are pinned, as these checkpoints are used to quickly establish a redo start point for the rollforward phase of crash recovery. Checkpoints are generated every minute, by default, meaning that at least two minutes worth of log records will be pinned on disk.
- LOGBACKUP has been enabled and inactive log files have not been backed up.

Log files become candidates for deletion under the following conditions:

- When the current log file becomes full and is rolled over or you force a log rollover. Note that a forced log rollover also occurs when you backup offline, and reset the NEXTLOG parameter by either a SQLTalk SET NEXTLOG command or a call to the *sqlset* API function, specifying the SQLPNLB parameter.
- When SQLBase unpins a log file by forcing a rollback on a transaction that has pinned the log file.
- When log files are backed up.
- If LOGBACKUP is off and the log is no longer needed for rolling back transactions or crash recovery.
- When database logging is turned off by a SQLTalk SET RECOVERY OFF command.

An attempt to delete unpinned log files is *not* made at every commit/rollback, as this would affect performance.

Log file deletion

If media recovery is important to your site, you will need to set the LOGBACKUP parameter on to specify that logs are to be backed up before they are deleted. Use the SQLTalk SET LOGBACKUP command or the *sqlset* API function.

The DBA is responsible for carrying out routine backup procedures at which time SQLBase automatically deletes log files. By default, LOGBACKUP is off, so that SQLBase deletes log files as soon as they are not needed to perform transaction rollback or crash recovery. This is done so that log files do not accumulate and fill up the disk.

If log files are backed up off-line, the NEXTLOG parameter needs to be set once the database is brought back online. This parameter alerts SQLBase to the fact that an off-line backup has occurred and that there are logs eligible for deletion.

Releasing a log file for backup or deletion

The SQLTalk RELEASE LOG command and the API function *sqlrel* force the release of the currently active log file, making it available for backup or deletion by SQLBase.

SQLBase closes the current log file and creates a new one automatically when the current log file becomes full (this is called log *rollover*). The RELEASE LOG command and the *sqlrel* function let you back up a log file without waiting until it becomes full. This allows you to get the most current backup possible.

The RELEASE LOG command is not needed if the BACKUP SNAPSHOT command was used to make the backup. BACKUP SNAPSHOT automatically forces a log rollover.

If LOGBACKUP is off and you do a RELEASE LOG or *sqlrel*, the log file will not be deleted until the *next* log rollover.

Turning recovery off

If a system failure occurs while recovery is off, your database will be in an inconsistent state and you can lose referential integrity.

You should only turn off recovery/logging if you can afford to lose your data in the event of a system crash or media failure. Use SQLTalk's SET RECOVERY OFF command (or connect using *sqlcnr*) to do this.

Once you set recovery off, the settings for autocommit, bulk execute, cursor-context preservation, restriction mode, rollback, scroll mode, isolation level, loadversion, and timeout will revert back to their defaults. Be aware that with recovery off, rollback and autocommit are meaningless.

You should back up the database and log files before you turn recovery off.

Temporary files

In the course of processing, SQLBase single-user servers and multi-user servers can create several kinds of temporary files:

- Sort files
- Read-only history files
- General-use files

SQLBase names temporary files *sqlxxxx.tmp* where *xxxx* is a randomly-generated serial number. It places temporary files in one of three places, in this order:

1. In the directory specified by the *tempdir* configuration keyword
2. In the directory specified by the *tempdir* environment variable
3. In the current directory

Sort files

Sort files contain the end result of sorts specified by a DISTINCT, ORDER BY, GROUP BY, or CREATE INDEX clause. SQLBase creates one sort file per sort clause.

A sort file is separate from a general-use file even if the data being sorted is part of the result set data contained in the general-use file. Once the result set is built in the general-use file, SQLBase deletes the sort file.

Read-only history files

Read-only history files contain multiple copies (before-images) of changed data pages. History files enable SQLBase to offer read-only transactions a consistent view of the data as it existed when the read-only transaction began. SQLBase creates one read-only history file per database.

Read-only transactions. Read-only transactions may affect performance, so they are disabled by default.

To override the default setting and enable read-only mode for all databases on a server, set the *readonly* configuration keyword value to 1. To override the default setting and enable read-only mode for a specific database, use SQLTalk's SET READONLY command or the SQL/API's *sqlset* function with the SQLPROD parameter.

History file size. If you have enabled read-only mode, you can specify the size of the history file in kilobytes with SQLTalk's SET HISFILESIZE command or the SQL/API *sqlset* function. The default history file size is 1000 kilobytes (1 megabyte).

Read Only isolation level. When read-only mode is on, you can use the Read Only (RO) isolation level. This isolation level provides a before-image view of the data as it was when the SELECT command was executed.

The Read Only isolation level places no locks on the database and you can only use it for reading data.

General-use files

General-use files contain result sets, temporary tables, temporary indexes used in processing joins, and hash join information. SQLBase creates one general-use file per database.

How SQLBase uses read-only history and general-use files

Historically, SQLBase created temporary files in pairs and used them in a cyclical manner. When the first file reached 1000 pages in capacity, SQLBase began writing to the second file. This was known as a rollover. When SQLBase determined that none of the transactions begun in the first file were needed any longer, it unpinned and deleted the file and created another in its place. Ideally, this happened before the second file reached 1000 pages in capacity. If not, the second file continued to grow.

Currently, SQLBase does not create temporary files in pairs. Instead, it creates one temporary file and recycles the space in that file. For example, if you are in result set mode, committing a transaction or compiling or executing another SQL statement in the same transaction frees the result set created by your *last* SQL statement. This allows SQLBase to release the space in the temporary file occupied by that result set, and enables another transaction to claim that space for its own processing. By recycling already-existing space in this way, SQLBase can prevent temporary files from growing too large.

In addition, SQLBase attempts to delete the temporary file whenever possible. If none of the transactions in the temporary file are needed any longer, SQLBase deletes it. For example, assume that you are the only user connected to a database and you are in result set mode. If you compile and execute a query, SQLBase builds a result set in the temporary file. If you then re-compile and re-execute another query, SQLBase frees the space allocated in the temporary file for your first result set and deletes the temporary file. The results of your second query will be put into a *new* temporary file.

If you are in result set mode and you set cursor-context preservation on for one of a transaction’s cursors, a commit (unlike a re-compile or re-execute command) does *not* free the result set. The space in the temporary file occupied by the result set remains claimed, preventing other transactions from using it.

If you are in restriction mode, the result sets created by each re-compile and re-execute are not freed. Neither are they freed when you issue a commit. That is because these result sets are needed to serve as the data sources for each successive query you issue.

Ultimately, if all existing space in the temporary file is claimed, SQLBase must increase the size of the temporary file to enable other transactions access to it.

Partitions

As databases grow, disk space becomes critical. SQLBase enables you to spread your databases across multiple disk partitions to handle larger databases and increase performance. SQLBase provides a way for you to use partitions on all available platforms. In this way, database size is only limited by the total amount of disk space available on the server.

SQLBase enables you to define multiple named *database areas*, which are physical files or devices where database data and log files can be stored. These areas can be spread across multiple disk volumes to take advantage of parallel disk input/output operations.

You can spread database log files across different devices or disks. This allows concurrent log backups.

What is a partition?

Partitioning maximizes disk space by dividing a disk into different database areas on multiple disks. Partitions can be on different disks on the same system; they *cannot* be on separate network computers. A database can extend across several partitions without slowing down performance. Performance may actually be increased by allowing simultaneous writes to the database, as well as by reducing the time to write to the database and log files.

Partitioning can be turned on and off with the following SQLTalk commands:

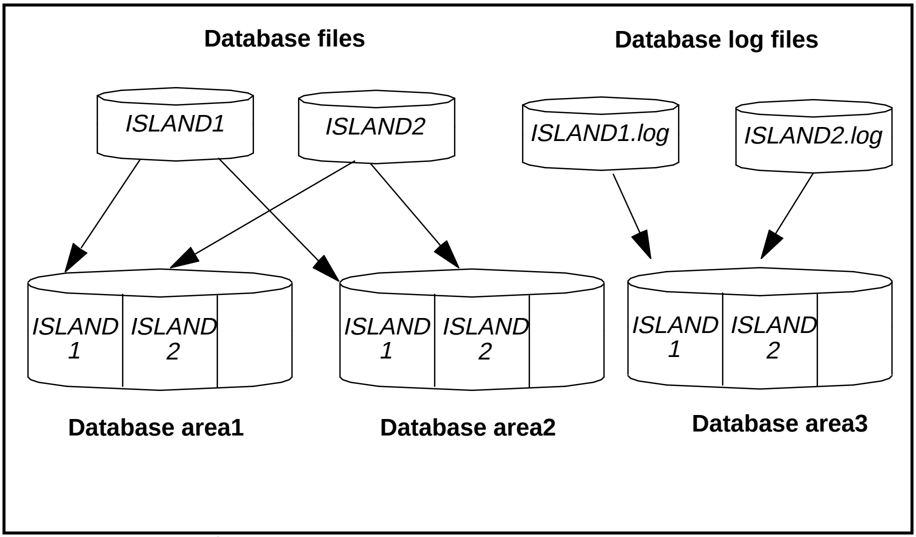
```
SET PARTITIONS ON;

SET PARTITIONS OFF;
```

You must disable partitioned databases before you restore the MAIN database from backup, and you must re-enable them once recovery is complete.

What are database areas?

A database area is a portion of a disk specified by the user for the storage of database and/or log files. A database area can reside either on a normal file system or a NetWare file server volume. You can specify many database areas of various sizes and give them names.

Disk space is distributed from a database area to a particular database in units called extents. An extension unit is the amount of contiguous disk space in a database area which is allocated to a database or log file when that file is extended. The default extension unit value is 1 megabyte but this can be set to higher multiples of 1 megabyte.



Databases and log files spread across three database areas

As you can see in the above figure, there are two sample ISLAND databases and log files spread across three database areas. You can accomplish this by creating the database areas, assigning them to storage groups, and then creating databases in the storage groups.

You can increase or decrease the allocation size from the default of 1 megabyte, by using either:

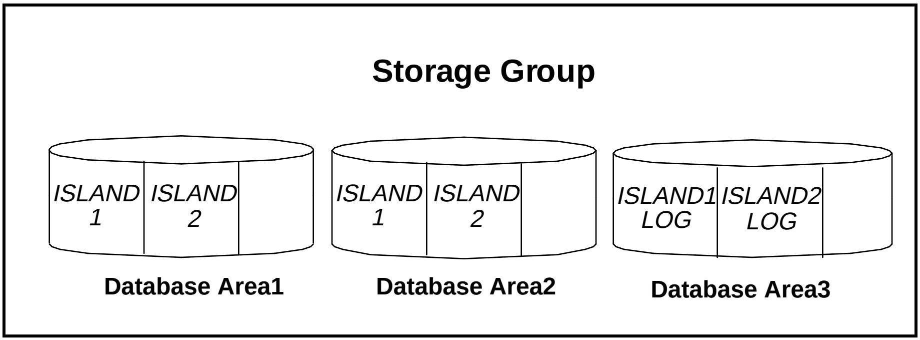
- SQLTalk's SET EXTENSION command.
- The SQL/API's *sqlset* function with the SQLPEXS parameter.

What are storage groups?

A storage group is a named list of database areas which is used to organize and specify partitioned databases. A similar concept is an electronic mail group, which is a named list of electronic mail recipients.

You can declare storage groups for many different functions. For example, you could declare a storage group for high-access databases that lists the areas from one or more fast disks. Another storage group could list areas for low-access databases which reside on slower disks. A database area can appear on multiple storage group lists.

Using storage groups allows you to store databases on different volumes. Storage groups also allow the database system to automatically handle most of the details of allocating space. In the following graphic, the three database areas form one named storage group.



Three database areas form a storage group

When you create a database, you can specify a storage group to designate where the file space for that database is located. You can also specify a default storage group to handle databases that do not explicitly designate a storage group. If you do not specify a default storage group, the database is created in the normal file system.

Each database area within a storage group may contain extents allocated to different databases. In addition to selecting a storage group for the database, you can specify a storage group for the logs of that database. This enables you to place the logs on a disk separate from the database for better performance and data integrity.

The MAIN database

SQLBase creates a MAIN database under either of the following conditions:

- If you create a partitioned database.
- If you enable commit service for a database by setting the *commitserver* keyword in the sql.ini file. Read *Chapter 9, Distributed Transactions* for information on commit service.

The MAIN database stores control information that you can query:

- Partitioned database file names
- Database area and extent names and sizes
- Storage group names
- Participants in a distributed transaction
- Two-phase commit operations

To connect to the MAIN database, specify the name of the server on which the MAIN database resides when you are prompted to enter the database name.

You should backup the MAIN database whenever you backup partitioned files. You can restore the MAIN database, but you cannot change its columns and rows.

You cannot create a database called MAIN yourself, nor should you have an existing database called MAIN. In addition, do not delete the MAIN database.

Read *Appendix A, System Catalog Tables* for a list of tables and views in the MAIN database.

MAIN's initialization file (main.ini)

The *main.ini* file is an initialization file used only for creating the MAIN database.

The *main.ini* file must be in the same directory as that from which you start the database server. This means that *main.ini* must be in the your current directory when starting in single-user mode and in the server software directory when in multi-user mode. In both cases, C:\Centura is the default.

The *main.ini* file creates the GUEST/GUEST account, as well as creates views, synonyms, and stored commands which enable you to access the partitioned database information contained in the MAIN database.

For example, to find out if a database is partitioned, connect to the server name and use the guest account to log on to MAIN. Then query the public synonym DATABASES. As another example, to get information about available space, you can query the public synonym FREEEXTS.

Read *Appendix A, System Catalog Tables* for a list stored commands contained in the *main.ini* file.

Database auditing

SQLBase's database auditing feature allows you to audit database activities and performance. For example, you can monitor who logs on to a database, which tables they access, and if they attempt to access data for which they do not have privileges. You can also measure server performance by gathering information on command execution time and long-running transactions. To gather this data, you start one or more *audit operations* that write output to an associated audit file.

You can have up to 32 active audit operations running concurrently. However, generally you do not need more than one or two, since you can record different types of information within each audit. Also, be aware that each additional audit operation can affect performance.

All concurrent active audits must have unique names.

Audit file

An audit operation writes its output to an *audit file*. This is a simple ASCII file that you can print or access through either an editor or SQLConsole.

The audit file collects various types of system and performance information depending on the audit type and category you choose. You can also direct a message to an audit file to document a system activity.

An audit file name has the format <*auditname*>.x, where ***x*** is an ascending value. For example, starting an audit operation with the name *myaudit* generates an output file called *myaudit.1*.

You can record information to one audit file of unlimited size, or to a series of sequential files that have a specified size. When a file reaches this defined size, SQLBase automatically generates a new file. For example, if you specify a maximum size of 100 kilobytes when you start the *myaudit* audit, SQLBase automatically creates and starts writing output to *myaudit.2* when *myaudit.1* reaches a size of 100 kilobytes.

Since the sequence of audit files is controlled by the file extension, you can theoretically have up to 1000 audit files for each individual audit (though this may not be practical for real-time use). The audit file extension sequence ranges from <*auditname*>*. 1* to <*auditname*>*.999*. After <*auditname*>*.999*, the next file is called <*auditname*>*.0*, and then wraps around to begin the sequence again with <*auditname*>*.1*.

You can automatically delete old audit files by telling SQLBase to keep only a certain number of files besides the current file. For example, if you specify a KEEP value of 2 for an audit called *myaudit*, (with the SQL START AUDIT command), SQLBase automatically deletes *myaudit.1* when it creates *myaudit.4*, since it can only keep two old files **(***myaudit.2* and *myaudit.3*).

Starting and stopping an audit

To start an audit, use the START AUDIT command. This command creates the following entry in the appropriate section of your configuration file (sql.ini):

```
AUDIT=[type],auditname,[directoryname],[size-integer],
[append-integer],[keep-integer],[OVERWRITE],x,...
```

For example:

```
AUDIT=GLOBAL,SECURITY,C:\CENTURA,1000,1,1,OVERWRITE,1,2,3
```

For descriptions of these parameters, read the START AUDIT command documentation in the *SQL Language Reference*. The ***x*** field represents an information category, which is described in the next section.

An audit remains active while the server is running. It stops when you shut down the SQLBase server, but restarts when you bring the server back up. To stop an audit operation, use the SQL STOP AUDIT command.

Types of audit operations

SQLBase supports two types of audit operations: *global* and *performance*. These two audit types record different types of information. You specify what type of information to record by indicating its category when you run START AUDIT.

Both audit types also automatically record the following general information in the audit file:

Category	Description
0	Start of audit.
998	Records when a message was issued to an audit file. Read ***Sending a message to an audit file*** on page *4-25*.
999	End of audit.

The next two subsections describe the global and performance audit categories, and what type of information each category records. For examples and detailed information about each category, read the following sections.

Global

Use this type of audit to monitor activities for the entire SQLBase system. This audit type provides global information for the server.

The following table shows the types of information you can monitor with a global audit, and their category numbers. Categories with an asterisk (*) contain important security information that you may want to record on a continuing basis.

.

Category	Data collected	Description
***1**	Rejected logons	Records unsuccessful login attempts. Useful to see if a user tried to access a restricted database.
***2**	Security violations	Tells you if a user tried to access data without proper privileges.
***3**	Valid logins/logoffs	Records all valid logons, telling you when a user first connected, and whether he/she disconnected. Use it to find out what users were logged on.
4	Valid connects/disconnects	Records CONNECT and DISCONNECT statements.
5	Database creates, drops, installs, and deinstalls	Records CREATE DATABASE, DROP DATABASE, INSTALL DATABASE, and DEINSTALL DATABASE commands.
6	Recovery operations	Records all ROLLBACK commands.
7	Backup and restore operations	Records all BACKUP and RESTORE commands.
8	Database Lock Manager deadlocks and timeouts	Records information about deadlocks and timeouts.
9	Table access information (queries)	Tells you which users accessed which tables.
10	Table update information (inserts, updates, and deletes)	Records database manipulation language commands (DML), and which users issued the commands.

Performance

Use this type of audit to monitor performance for a particular operation. A performance audit records how long a SQLBase operation takes, and is useful for performance analysis and tuning.

A performance audit records the performance of the following operations:

Category	Data collected	Description
101	Connects and Disconnects	Tells you how long it takes users to connect to and disconnect from a database.
102	SQL command compilation, execution, storage, and retrieval.	Tells you how much time SQLBase takes to compile, store, retrieve, and execute a particular SQL statement.
103	End of transaction.	Tells you when a transaction ended, and how long the transaction took. Useful for locating long-running transactions.

Record formats

This section provides the syntax of the different audit categories as they appear in the audit file. It also provides examples of the general record layouts: START AUDIT (category 0) and STOP AUDIT (category 999).

Many of the record layouts contain one or more of the following general fields:

Field Names	Description	Format
datetime	The date and time the information for this record was recorded.	***dd-mon-yy hh:mi:ss.99*** Read the *SQL Language Reference* for information on this date/time format.
database	The name of the database	Database name, without the ***.dbs*** extension. For example: ISLAND
clientname	The name that is displayed in the CLIENT NODE field on the SERVER STATUS display.	Client name; for example: WinUser
elapsedtime	This field appears in the performance audit file. It shows the amount of time SQLBase took to complete the activity.	Depending on the length of time the activity takes, this field can appear in one of three formats: *ss.99* *mi:ss.99* *hh:mi:ss.99* Read the *SQL Language Reference* for information on these date/time formats.

START AUDIT and STOP AUDIT (categories 0 and 999)

This section shows examples of the START AUDIT and STOP AUDIT records written to the audit file.

Record	Format
Start global audit record (category 0)	000,datetime,GLOBAL,auditname,x,x,x... where ***x*** represents an audit category specified in the START AUDIT command.
Start performance audit record (category 0)	000,datetime,PERFM,auditname,x,... where ***x*** is an audit category specified in the START AUDIT command.
Stop audit record (category 999)	999,datetime,auditname

Global audit categories

This section shows examples of all global audit categories.

Record	Format
Rejected logon (category 1)	001,datetime,error#,database,username,clientname
SQL security violation (category 2)	002,datetime,error#,database,username,clientname,tablename
Valid logon/logoff (category 3)	003,datetime,LOGON,database,username,clientname 003,datetime,LOGOFF,database,username,clientname
Valid CONNECTS/ DISCONNECTS (category 4)	004,datetime,CONNECT,database,username,clientname 004,datetime,DISCONNECT,database,username,clientname
CREATE, DROP, INSTALL, and DEINSTALL DATABASE (category 5)	005,datetime,CREATE database, 005,datetime,DROP database 005,datetime,INSTALL database 005,datetime,DEINSTALL database
Recovery operation (category 6)	006,datetime,database,username,clientname

Record	Format
BACKUP and RESTORE operations (category 7)	007,datetime,BACKUP DATABASE,database,username,clientname 007,datetime,BACKUP LOGS,database,username,clientname 007,datetime,BACKUP SNAPSHOT,database,username,clientname 007,datetime,RELEASE LOG,database,username,clientname 007,datetime,RESTORE DATABASE,database,username,clientname 007,datetime,RESTORE SNAPSHOT,database,username,clientname 007,datetime,ROLLFORWARD CONTINUE,database,username, clientname 007,datetime,ROLLFORWARD END,database,username,clientname 007,datetime,ROLLFORWARDSTART,database,username,clientname
Database lock manager deadlocks and timeouts (category 8)	008,datetime,DEADLOCK,database,user,clientname,xx,nnn,group#, page#,lock type, SQL STATEMENT... 008,datetime,TIMEOUT,database,user,clientname,xx,nnn,group#, page#,lock type, SQL STATEMENT... 008,datetime,LOCK HOLDER,database,user,clientname,xx,nnn,group#,page#,lock type, SQL STATEMENT... The ***xx*** field is one of the following isolation levels: • *RR* (Read Repeatability) • *CS* (Cursor Stability) • *RL* (Release Locks) • *RO* (Read-only) The ***nnn*** field is the Lock Timeout Value expressed in number of seconds. The following list shows the lock types: • ***Temp S-Lock*** (temporary shared lock) • ***S-Lock*** (shared lock) • ***Temp U-Lock*** (temporary update lock) • ***U-Lock*** (update lock) • ***Temp X-Lock*** (temporary exclusive lock) • ***X-Lock*** (exclusive lock) • ***Increment*** Whenever a TIMEOUT or DEADLOCK occurs, the audit displays the SQL statements for each lock holder involved. The user name resulting in a timeout or deadlock is printed last.

Record	Format
Table access information (queries only) (category 9)	009,datetime,database,username,clientname,tablename
Table update information (INSERTS, UPDATES, AND DELETES) (category 10)	010,datetime,database,username,clientname,INSERT INTO tablename 010,datetime,database,username,clientname,UPDATE tablename 010,datetime,database,username,clientname,DELETE FROM tablename

Performance audit category examples

This section shows examples of all the performance audit categories.

Record	***Format***
Connects and disconnects (category 1)	001,datetime,CONNECT,database,username,clientname,elapsedtime 001,datetime,DISCONNECT,database,username,clientname, elapsedtime
SQL Command compilation, execution, retrieval, and storage (category 2)	002,datetime,COMPILE,database,username,clientname,elapsedtime SQL STATEMENT... 002,datetime,EXECUTE,database,username,clientname,elapsedtime SQL STATEMENT... 002,datetime,RETRIEVE,database,username,clientname,elapsedtime SQL STATEMENT... 002,datetime,STORE,database,username,clientname,elapsedtime SQL STATEMENT...
End of transaction (category 3)	003,datetime,COMMIT,database,username,clientname,elapsedtime 003,datetime,ROLLBACK,database,username,clientname, elapsedtime

Sending a message to an audit file

You can include a message string in an audit file. For example, you may wish to document when you issue a COMMIT in an application, or when a program is waiting for a user response. The message can contain up to 254 characters.

To include a message in an audit, use the AUDIT MESSAGE command. Specify the message and an existing audit operation name. SQLBase writes the message in the audit file(s) associated with this audit name. You can send a message to the audit files for one or all audit operations.

An audit message is indicated by a 998 category number in the audit file, and uses the following record layout:

```
998,datetime,database,username,clientname,audit message
```

The following example shows a sample audit message record in the audit file:

```
998,23-SEP-95 17:00:23.17,ISLAND,SYSADM,WinUser,Example AUDIT
message
```

Audit file examples

The following examples show both global and performance audit files.

Global audit

This example shows a sample global audit file.

```
Start global audit  →  000,09-DEC-95 10:47:05.75,GLOBAL,TEST,1,2,3,4,5,6,7,8,9,10
                       004,09-DEC-95 10:47:05.86,CONNECT,ISLAND,SYSADM,
                       004,09-DEC-95 10:47:05.86,DISCONNECT,ISLAND,SYSADM,
                       001,09-DEC-95 10:47:05.92,404,ISLAND,DAVID,
                       001,09-DEC-95 10:47:06.03,404,ISLAND,SYSADM,
                       010,09-DEC-95 10:47:06.47,ISLAND,SYSADM,,INSERT INTO SYSADM.X
                       004,09-DEC-95 10:47:06.52,CONNECT,ISLAND,DAVID,
                       002,09-DEC-95 10:47:06.58,1102,ISLAND,DAVID,,X
                       005,09-DEC-95 10:47:12.34,CREATE TEST
                       005,09-DEC-95 10:47:12.40,DEINSTALL TEST
                       005,09-DEC-95 10:47:12.40,INSTALL TEST
                       005,09-DEC-95 10:47:12.40,DROP TEST
                       006,09-DEC-95 10:47:12.89,ISLAND,SYSADM,
                       010,09-DEC-95 10:47:12.95,ISLAND,SYSADM,,UPDATE SYSADM.X
                       009,09-DEC-95 10:47:13.00,ISLAND,SYSADM,,SYSADM.X
                       010,09-DEC-95 10:47:13.06,ISLAND,SYSADM,,DELETE FROM SYSADM.X
Trace message       →  998,09-DEC-95 10:47:13.06,ISLAND,SYSADM,,This is The End
Stop this audit     →  999,09-DEC-95 10:47:13.11,TEST
```

Performance audit

This example shows a sample performance audit file.

The End of Transaction (category 103) marks a COMMIT or ROLLBACK statement. If there is only one user on the system, such as in the following example, all of the category 1 and 2 records contained between the category 103 records are part of the same transaction. However, if there are multiple users on the system, these records are not necessarily part of the same transaction since they can be generated by different users.

Note that the time elapsed field for the category 103 record records the elapsed time since the last category 103 record, not the sum of the elapsed times of the preceding category 101 or 102 records.

```
Start performance      →  000,30-NOV-95 16:57:59.20,PERFM,perf,1,2,3
audit                     101,30-NOV-95 16:58:10.02,CONNECT,ISLAND,SYSADM,
                            internal operation,0.20
End of transaction 1   →  101,30-NOV-95 16:58:10.08,DISCONNECT,ISLAND,SYSADM,internal
                            operation,0.00
                          103,30-NOV-95 16:58:21.61,ISLAND,SYSADM,,10.75
                          101,30-NOV-95 16:58:21.67,CONNECT,ISLAND,SYSADM,,1.00
End of transaction 2.  →  103,30-NOV-95 16:58:23.97,ISLAND,SYSADM,,2.10
Note the elapsed time     102,30-NOV-95 16:58:24.14,COMPILE,ISLAND,SYSADM,,0.10,
of 2.10 is the amount     UPDATE X SET A = 1
of time elapsed since     102,30-NOV-95 16:58:48.36,EXECUTE,ISLAND,SYSADM,,22.05,
transaction 1 ended.      UPDATE X SET A = 1
                          103,30-NOV-95 16:58:48.64,ISLAND,SYSADM,,22.40
Trace message          →  998,30-NOV-95 17:15:17.86,SYSADM   ANOTHER COMMIT
                          102,30-NOV-95 17:15:17.86,EXECUTE,ISLAND,SYSADM,,0.00,
                          103,30-NOV-95 17:15:33.35,COMMIT,ISLAND,SYSADM,,29.90
Stop this audit        →  999 23-SEP-95 17:03:33.99,perf
Start new audit        →  100,23-SEP-95 17:00:13.84,PERFM,perf2,1,2,3
                          .
                          .
                          .
```

Chapter 5

DBA Interfaces

This chapter introduces the client programs that you as a DBA can use to administer databases.

Details on both the character and GUI interfaces of the servers are also included.

Client programs

This section describes Centura's client programs which enable you to perform DBA tasks and maintain SQLBase databases.

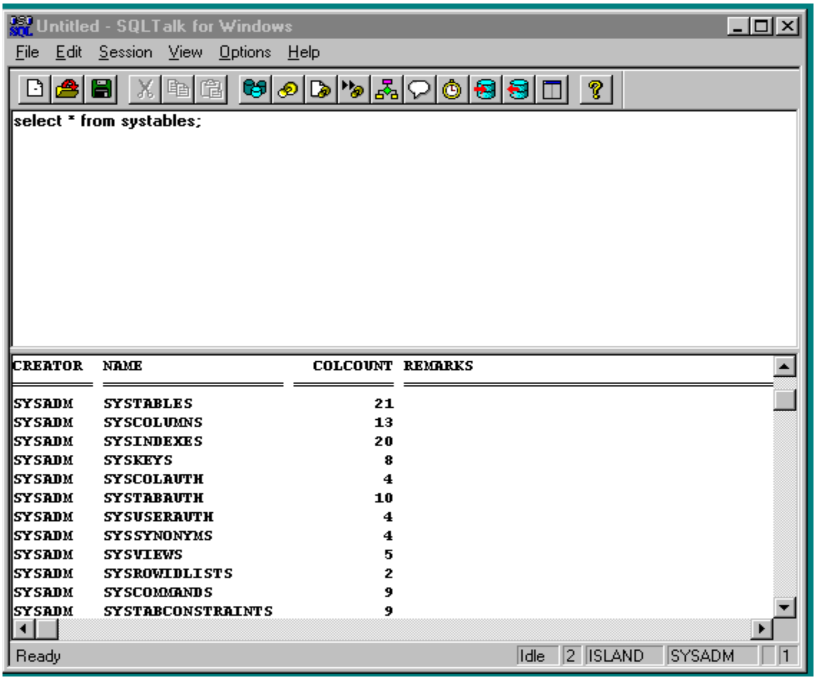
SQLTalk

SQLTalk is an interactive user interface for SQLBase, offering not only a complete implementation of SQL, but many extensions as well.

SQLTalk runs on the following client platforms:

- Windows 3.x using the executable s*qltlk16.exe*
- Windows 95 and Windows NT, both using the executable *sqltalk.exe*.

The following example is an excerpt from a SQLTalk session on the Windows NT platform.



```
select * from systables;
```

```
CREATOR   NAME                     COLCOUNT REMARKS
========  =======================  ======== ===================================
SYSADM    SYSTABLES                      21
SYSADM    SYSCOLUMNS                     13
SYSADM    SYSINDEXES                     20
SYSADM    SYSKEYS                         8
SYSADM    SYSCOLAUTH                      4
SYSADM    SYSTABAUTH                     10
SYSADM    SYSUSERAUTH                     4
SYSADM    SYSSYNONYMS                     4
SYSADM    SYSVIEWS                        5
SYSADM    SYSROWIDLISTS                   2
SYSADM    SYSCOMMANDS                     9
SYSADM    SYSTABCONSTRAINTS               9
```

Note that entering **CONNECT 1** causes SQLTalk to accept the default value of DEMO for the database, SYSADM for the username, and SYSADM for the password.

For more information about SQLTalk, refer to the *SQLTalk Command Reference* or the SQLTalk for Windows Online Help System.

Application Programming Interface (API)

Centura's API has a set of functions that you can call to access a database using Structured Query Language (SQL).

SQL (Structured Query Language) can define, manipulate, control, and query data in a relational database. However, SQL is not a programming language and does not provide:

- Procedural logic
- Extensive data typing
- Variables

The API is a language interface that lets you develop a client application that uses SQL. You embed SQL/API functions within your C program, and this enables you to use SQL without giving up the power and flexibility of the programming language.

You can use the SQL/API to write programs that perform DBA operations. This lets you create custom DBA client applications tailored to your specific needs.

The SQL/API is available for all client platforms. Read the *SQL Application Programming Interface Reference* for more information about the SQL/API.

SQLConsole

SQLConsole for SQLBase is a remote database server tool to perform monitoring and administration tasks. If you are a Windows user, you can use SQLConsole to perform most DBA operations described in this manual for a local SQLBase server or any SQLBase server on your network.

From a single Windows desktop, SQLConsole provides an easy-to-use graphical interface that lets you perform DBA operations without the need for SQL commands. SQLConsole simplifies basic and routine administration tasks, such as creating, installing, deleting, or backing up databases and log files.

SQLConsole offers these capabilities for monitoring and controlling databases and servers:

- Presents important lock, database, process, cursor and activity statistics to the database administrator or SQLBase developer.
- Provides access to a set of very powerful SQLBase functions previously available only to the advanced C programmer.

  For example, you may disconnect active sessions, shutdown a database gracefully or terminate a server.

- Provides access to SQLTalk for Windows custom control, an interactive interface for entering SQL and SQLTalk commands, as well as SQLTalk scripts.

As a powerful ally in your database administration SQLConsole also provides these managers:

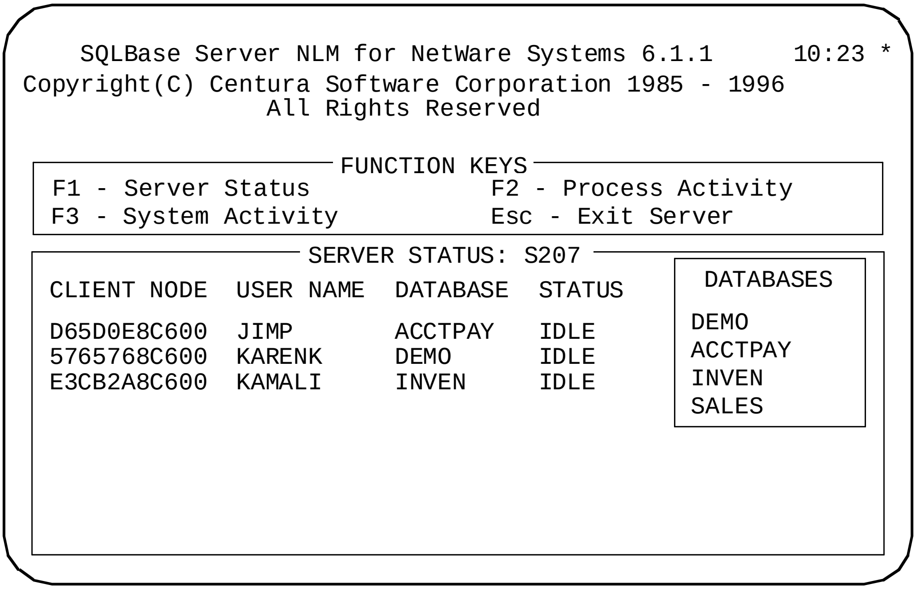
- ***Scheduling Manager***: automates your mission-critical backup cycle.
- ***Database Object Manager***: accesses all facets of a server through is graphical user interface.
- ***Alarm Manager***: alerts you on state of your server.

Read the *SQLConsole Guide* for details on using SQLConsole for daily database administration and development activities.

Character-based server interface

The SQLBase server for NetWare has three character-based displays that let you monitor server activity. You can switch among the displays using the **F1**, **F2**, and **F3** function keys.

Server Status display screen (F1)



```
    SQLBase Server NLM for NetWare Systems 6.1.1     10:23 *
Copyright(C) Centura Software Corporation 1985 - 1996
                    All Rights Reserved

 ┌──────────────────────FUNCTION KEYS──────────────────────┐
   F1 - Server Status              F2 - Process Activity
   F3 - System Activity            Esc - Exit Server
 └─────────────────────────────────────────────────────────┘
 ┌────────────────────SERVER STATUS: S207────────────────────┐
   CLIENT NODE  USER NAME  DATABASE  STATUS     DATABASES

   D65D0E8C600  JIMP       ACCTPAY   IDLE      DEMO
   5765768C600  KARENK     DEMO      IDLE      ACCTPAY
   E3CB2A8C600  KAMALI     INVEN     IDLE      INVEN
                                               SALES
```

Server Status display of a SQLBase Server for NetWare

Banner. The banner tells you the name and version of the product currently running — in this case, SQLBase Server for NetWare, version 6.1.1.

Function Keys. The function keys section serves as a guide for getting to the rest of the displays and bringing down the server.

Server Status. By default, SQLBase randomly generates a name for the server (in this case, S207) if you do not configure a *servername* keyword in the server's configuration file (sql.ini).

The Server Status section lists both the databases on the server and the clients connected to those databases:

Client Node	Identifies the client node associated with this process. This first appears as a 0 (zero) and then changes to an identifying number once the connection is successful. Based on the platform, SQLBase has various ways of generating this value: For a Windows client, SQLBase displays a randomly-generated number. For a client whose configuration file (sql.ini) includes a *timebased* keyword, SQLBase displays a timestamp. If a client connecting to the server has a *clientname* keyword configured in its sql.ini file, that value overrides any of the above defaults. Read *Chapter 3, Configuration (sql.ini)* for more information about the *clientname* keyword.
User Name	The name of the user associated with this process.
Database	The database being accessed.
Status	Whether the process is active or idle. Active means that the process is in the server's run queue.

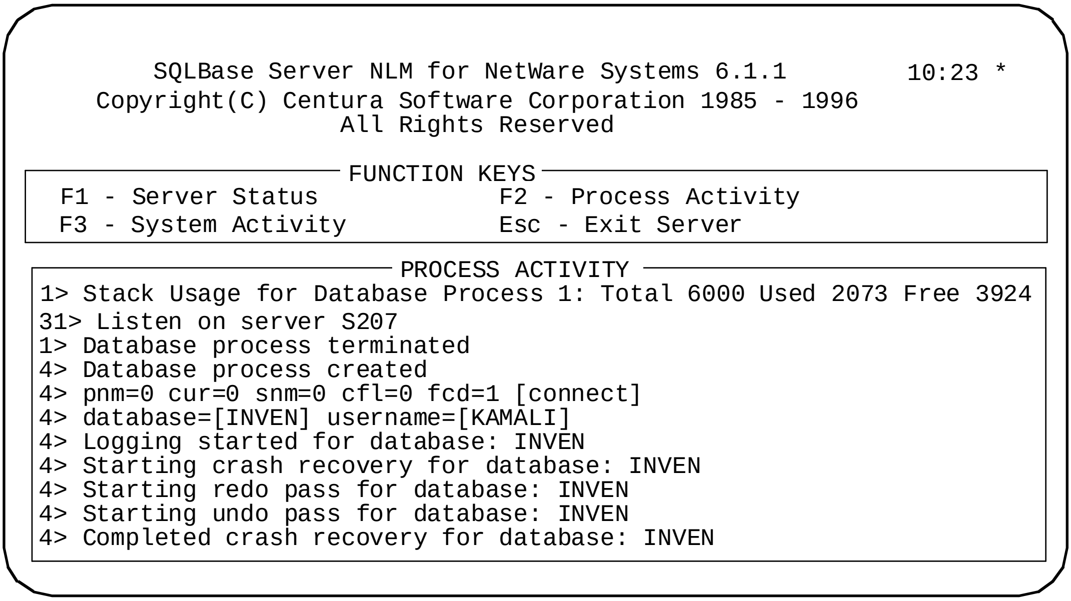
By default, the server displays only the Client Node, User Name, and Database information. You can press the plus (+) key on the keypad to display the Status column, although this may affect system performance. Press the minus (-) key to remove the Status column.

Databases. This window lists the databases on the server. SQLBase gets this information from the *dbname* keywords in the configuration file (sql.ini).

Busy Indicator. The server displays a busy indicator (flashing asterisk) in the top right corner of the screen when the server is active. When the busy indicator is *not* displayed, the server is idle. The asterisk is not displayed at print level 0 (see next section).

SQLBase updates the Server Status display every time a client connects to, or disconnects from, a database. When clients are not frequently connecting and disconnecting, SQLBase refreshes the display periodically. The refresh frequency decreases over time if the server is idle. Because of this, the displayed time may not always be accurate.

Process Activity display screen (F2)



```
        SQLBase Server NLM for NetWare Systems 6.1.1        10:23 *
    Copyright(C) Centura Software Corporation 1985 - 1996
                     All Rights Reserved

                         FUNCTION KEYS
  F1 - Server Status              F2 - Process Activity
  F3 - System Activity            Esc - Exit Server

                         PROCESS ACTIVITY
1> Stack Usage for Database Process 1: Total 6000 Used 2073 Free 3924
31> Listen on server S207
1> Database process terminated
4> Database process created
4> pnm=0 cur=0 snm=0 cfl=0 fcd=1 [connect]
4> database=[INVEN] username=[KAMALI]
4> Logging started for database: INVEN
4> Starting crash recovery for database: INVEN
4> Starting redo pass for database: INVEN
4> Starting undo pass for database: INVEN
4> Completed crash recovery for database: INVEN
```

Process Activity display of a SQLBase Server for NetWare

Banner. The banner tells you the name and version of the product currently running — in this case, SQLBase Server for NetWare, version 6.1.1.

Function keys. The function keys section serves as a guide for getting to the rest of the displays and bringing down the server.

Process Activity. The Process Activity display shows information about the tasks the server is performing.

If you have set your message display at level 2 or higher, the database processes information displays the following parameter values:

Field	Explanation
PNM	The client process number.
SNM	The client communication session number.
CFL	A flag byte value
FCD	Function code

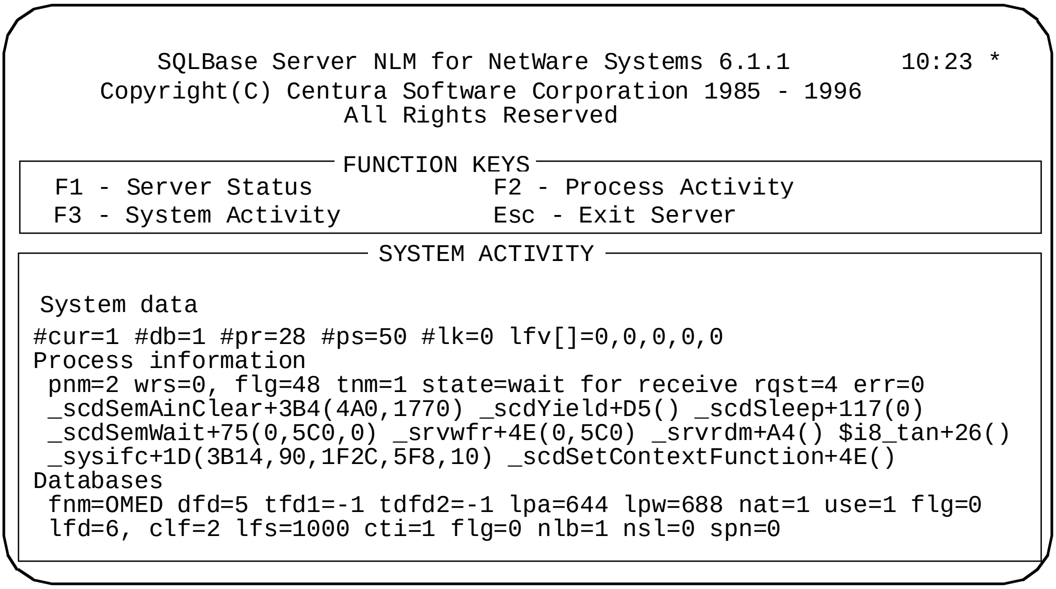
Message levels. Press the plus (+) key on the keypad to display additional levels of messages. The minus (-) key returns you to the previous level.The five message levels are:.

Message Level	Description
0	The installation of the server and database names on the network, as well as each database process's creation and termination.
1	All the information available at level 0 plus SQL statements and error numbers. (Note this includes viewing messages for the LOAD and UNLOAD commands.)
2	All the information available at level 1 plus the compile and execute processing steps, the ability to see the processing of individual cursors within a database process, and internal processing information.
3 and 4	All the information available at level 2 plus the fetch processing step and bind variable data.

Logging. Configure the *log* keyword in the server’s configuration file (sql.ini) to write the Process Activity messages to a file. These messages are helpful during system development, but they are not meant to be logged in a production system. If you are logging database messages, the log file can get quite large. Also, the server has to do additional work to write to the log file which affects system performance.

If you press **Alt-T**, SQLBase timestamps each message. Press **Alt-T** again to toggle off the timestamping feature.

System Activity display screen (F3)



```
          SQLBase Server NLM for NetWare Systems 6.1.1       10:23 *
      Copyright(C) Centura Software Corporation 1985 - 1996
                         All Rights Reserved

                          FUNCTION KEYS
  F1 - Server Status              F2 - Process Activity
  F3 - System Activity            Esc - Exit Server

                            SYSTEM ACTIVITY

 System data
#cur=1 #db=1 #pr=28 #ps=50 #lk=0 lfv[]=0,0,0,0,0
Process information
 pnm=2 wrs=0, flg=48 tnm=1 state=wait for receive rqst=4 err=0
 _scdSemAinClear+3B4(4A0,1770) _scdYield+D5() _scdSleep+117(0)
 _scdSemWait+75(0,5C0,0) _srvwfr+4E(0,5C0) _srvrdm+A4() $i8_tan+26()
 _sysifc+1D(3B14,90,1F2C,5F8,10) _scdSetContextFunction+4E()
Databases
 fnm=OMED dfd=5 tfd1=-1 tdfd2=-1 lpa=644 lpw=688 nat=1 use=1 flg=0
 lfd=6, clf=2 lfs=1000 cti=1 flg=0 nlb=1 nsl=0 spn=0
```

System Activity display of a SQLBase Server for NetWare

Banner. The banner tells you the name and version of the product currently running — in this case, SQLBase Server for NetWare, version 6.1.1.

Function keys. The function keys section serves as a guide for getting to the rest of the displays and bringing down the server.

System Activity. The System Activity display shows system information about internal server activity.

The System Activity display has three sections:

- System data
- Process information
- Databases.

System data

This section contains information about the server.

Field	Explanation
cur	The number of cursor slots allocated since the server started.
db	The number of activated databases since the server started.
pr	The maximum number of users allowed on the server. You can set this limit in the server's sql.ini file with the *users* keyword. It can be higher than the value specified because of system overhead.
ps	The number of cache pages on the server. You can set this value in the server's sql.ini file with the *cache* keyword.
k	The number of lock entries on the server. You can set this value in the server's sql.ini file with the *locks* keyword. The default value of 0 means there is no limit on the number of locks.
fv	The system resources held. A non-zero value means that the resource is being held and represents the usage; a value of zero means that the resource is not being held.

Process information

This section contains information about each process:

Field	Explanation
nm	The process number.
wrs	The resource for which the process is waiting.
lg	Flags.
nm	The transaction number.
state	A message indicating the process's activity, such as: "waiting for request" "performing request" "sending response" "handling error" "wait for IO lock" "wait for page" "request done" "waiting for received" (waiting to receive something from the client)
qst	The last request received.

Field	Explanation
err	The return code of the last request.

It then lists the system locks held by the current process:

Code	Explanation
ST	A single thread system lock.
KI	A keyboard input system lock.
CNC	A connect/disconnect system lock.

Is also shows the system locks that the current process is for. This part is absent if the process is not waiting for a system lock.

Finally, it shows the resource (page) the current process is waiting for. This part is absent if the process is not waiting for a resource:

Field	Explanation

wlm	The lock mode the current process is requesting:
	S = Shared
	U = Update
	X = Exclusive
	I = Increment
	When a T is prefixed to S, U, or X, it signifies a temporary lock.

wdb	The database number.
wgp	The group number of the page.
wpg	The page number.

Databases

This section contains information about the databases that the server listens on:

Field	Explanation
nm	The database name.
dfd, fd1, dfd2	The file descriptor.
pa	The last page allocated.

Field	Explanation
lpw	The last page written (the database file size in pages).
nat	The number of active transactions.
use	The number of connected cursors.
lg	Flags.
fd	The log file descriptor.
clf	The current log file number.
lfs	The log file size in kilobytes.
cti	The checkpoint time interval in minutes.
flg	Flags.
nlb	The next log to backup.
nsl	The next log to backup (after a BACKUP SNAPSHOT).
spn	The transaction span limit (0 means no limit).
cpp, cpl ,ftp ,fap ,fbl, lbl	Log pointers.

The last part of the Databases section is a dump of the lock table. For example:

```
    1  S004:006:006
```

The ‘1’ indicates the transaction number that owns the lock entry. In some cases, this is the process number for special locks.

The ‘S’ means that it is a shared lock. This can also be:

X = Exclusive
U = Update
I = Increment

When T is prefixed to S, U, or X, it signifies a temporary lock.

The ‘004’ is the group to which the lock belongs. SQLBase divides pages in a database into groups.

The first ‘006’ is the starting page number and the second ‘006’ is the ending page number.

NetWare specifics

If the file server is not currently displaying one of the SQLBase server display screens, press **Ctrl** and **Esc** and select SQLBase from the task list.

GUI Server Interface

This section describes the Graphical User Interface (GUI) to SQLBase Servers for Windows 95 and Windows NT.

SQLBase Server

This is the main window that appears when you start the database server. It contains four sub-windows:

- Server Status
- Databases
- Process Activity
- System Activity

These sub-windows are very similar to child windows in an MDI application in that you can manipulate them individually. You can resize, move, open or close, or minimize or maximize one of them, all of them, or any combination of them. They must always reside within the main window though.

These windows are described in the sections following the next subsections on the SQLBase Server window's menus.

Saving configuration information

If you manipulate any of the server windows to a customized setting, you can save this configuration. You can also save specific display levels and timestamp settings. To save these configurations, use the *Save Settings* option of the server's **File** menu.

Scrolling and pausing

You can scroll or pause the information displayed on any of the sub-windows. Be aware that scrolling does not prevent SQLBase from writing new output to a sub-window. For example, if you are scrolling backwards and SQLBase has new output to write to the display, it will scroll forward to the end of the output and append the new information to the display.

To prevent SQLBase from writing new output to a window and interrupting you while scrolling, select the **Pause** menu command, and then select the name of the window to pause. A check mark indicates that the window is paused.

When you are finished scrolling, select the name of the window to un-pause from the **Pause** menu command again. The check mark disappears, and output resumes.

Because all of the sub-windows (except the Process Activity window) display snapshots of the server's status at any particular time, when you discontinue **Pause**, SQLBase updates the display with the latest information.

However, the Process Activity window displays a log, not a snapshot. It contains a record of the server's activity over time. When you pause the display of this window, SQLBase buffers the output until you click on **Pause** again; this un-pauses the displays. The limit to the amount of information that SQLBase can buffer is 500 lines. When the buffer becomes full, SQLBase 'wraps' back to the beginning of the buffer and starts overwriting data. This means that the data being overwritten is lost. If you need to track large amounts of Process Activity data, you should log the data to a file using the **File** menus' **Open Log** menu command.

There are six menus on the SQLBase Server Display:

- File
- Display
- Pause
- Level
- Window
- Help

The following paragraphs describe these menus.

File

The **File** menu contains commands that control logging, sql.ini configuration settings, and exiting the server:

Open Log	Opens a file and begins logging the Process Activity display to this file. This command presents you with the *File Open* dialog box that lets you select an existing file, create a new file, or accept the default of *sqllog.txt* in the current directory. If the file you specify already exists, SQLBase asks you to confirm that you want to overwrite the file. This command is equivalent to configuring the *log* keyword in the server's sql.ini file, or using SQLTalk's SET ACTIVITYLOG command.
Close Log	Closes the process activity log file.

Save Settings	Saves the current GUI server-related settings to the server's sql.ini file. These settings include the: • Position and size of the main window and all sub-windows • Display level • Timestamp option setting You can also use this option to save the PM server screen appearance between server launches.
Clear Settings	Removes any GUI server-related settings from the server's sql.ini file. The next time you bring up the server, it uses the system defaults.
Exit	Exits SQLBase Server. If there are users connected to databases on the server, you are asked to confirm that you want to shut down the server. If you shut down the server while users are connected, all transactions in progress are rolled back.

Display

The **Display** menu contains the following commands that control the display of the sub-windows. If a sub-window is displayed, a check mark appears next to it on the menu.

Server Status	Toggles the display of the Server Status window on and off.
Databases	Toggles the display of the Databases window on and off.
Process Activity	Toggles the display of the Process Activity window on and off.
System Activity	Toggles the display of the System Activity window on and off.
All	Displays all of the currently-closed windows.
None	Closes all of the currently-displayed windows.

Pause

The Pause menu controls whether each subwindow is paused. If a subwindow is paused, a check mark appears next to it on the menu.

Level

The **Level** menu contains commands that control the level of detail displayed on the Process Activity window. Only one of the five levels (0-4) can be active at a time. The currently-active level is indicated by a check mark next to it on the menu. These menu selections are equivalent to using SQLTalk's SET PRINTLEVEL command.

If timestamps are enabled, a check mark appears next to the corresponding menu selection. This menu selection is equivalent to using SQLTalk's SET TIMESTAMP ON/OFF command.

Level 0	Displays the installation of the server and database names on the network, as well as each database process's creation and termination.
Level 1	Displays all the information available at level 0 plus SQL statements and error numbers.
Level 2	Displays all the information available at level 1 plus the compile and execute processing steps, the ability to see the processing of individual cursors within a database process, and internal processing information.
Level 3 & Level 4	Displays all the information available at level 2 plus the fetch processing step and bind variable data.
Increase Level	Increases the display level.
Decrease Level	Decreases the display level.
Show Timestamps	Toggles the display of timestamps on the Process Activity window on and off.

Window

The **Window** menu provides options for displaying open sub-windows:

Tile	Displays the sub-windows side-by-side so that the maximum amount of information is displayed.
Cascade	Displays the sub-windows layered on top of one another so that only their title bars are visible (except the top-most one).

Help

The **Help** menu provides the following information:

About SQLBase	Provides information about the server program: its name, version number, and copyright. It is equivalent to an *About* box.

Server Status

This sub-window displays the server name on the top line, and contains the following information for each connection:

Client Node	Identifies the client node. This first appears as a 0 and then changes to an identifying number when the connection is successful: • For a Windows client application, this is a randomly-generated number. • If a client communicating via NetBIOS has a *timebased* keyword configured, this is a timestamp. • If a client has a *clientname* keyword configured, that value displays instead of any of the above.
User Name	Identifies the name of the user.
Database	Identifies the database to which each client is connected.
Status	Optional. You can press **Ctrl+** to display this field which shows you if the client is active or idle. Displaying the Status field can affect system performance, so press **Ctrl-** to remove this field from view when you do not need to see it. A status of ACTIVE means that the client process is in the server's run queue. Otherwise, the status displays as IDLE.

SQLBase updates the display every time a client node connects or disconnects.

Databases

This sub-window lists the databases on which the server listens. The database server gets this information from the *dbname* keywords configured in its sql.ini file.

Process Activity

This sub-window displays information about tasks that the server is performing.

Press **Ctrl+** to display additional levels of messages. Press **Ctrl-** to reduce the level of messages displayed.

See the **Level** menu explanation for detailed information about each level of display.

Configuring the *log* keyword in the server's sql.ini file, or selecting the **File** menu's **Open Log** command, writes the information displayed in this window to a specified file. This information is helpful during system development, but it is not meant to be logged in a production environment. Logging incurs overhead, and the log file can become large very quickly if your site is an active one.

System Activity

This sub-window displays system information about internal activity. The display has three sections:

- System data
- Process information
- Databases

System Data

This section contains data about the server:

Field	Description
cur	The number of cursor slots allocated since the server started.
db	The number of activated databases since the server started.
pr	The maximum number of users allowed on the server. You can set this limit in the server's sql.ini file with the *users* keyword. It can be higher than the value specified because of system overhead.
ps	The number of cache pages on the server. You can set this value in the server's sql.ini file with the *cache* keyword.
lk	The number of lock entries on the server. You can set this value in the server's sql.ini file with the *locks* keyword. The default value of 0 means that there is no limit on the number of locks.
lfv	The system resources held. A non-zero value means that the resource is being held and represents the usage; a value of zero means that the resource is not being held.

Process Information

This section contains the following information about each process:

Field	Description
pnm	The process number.
wrs	The resource the process is waiting for.
flg	Flags.
tnm	The transaction number.
state	A message indicating the process' activity, such as: "waiting for request" "performing request" "sending response" "handling error" "wait for IO lock" "wait for page" "request done" "waiting for received" (waiting to receive something from the client)
rqs	The last request received.
err	The return code of the last request.

It then lists the system locks held by the current process:

Code	Description
ST	A single thread system lock.
KI	A keyboard input system lock.
CNC	A disconnect system lock.

It also shows the system locks that the current process is waiting for. This part is absent if the process is not waiting for a system lock.

Finally, it shows the resource (page) the current process is waiting for. This part is absent if the process is not waiting for a resource:

Field	Description

wlm	The lock mode the current process is requesting:
	S = Shared
	U = Update
	X = Exclusive
	I = Increment
	When a T is prefixed to S, U, or X, it signifies a temporary lock.

wdb	The database number.
wgp	The group number of the page.
wpgw	The page number.

Databases

This section contains information about the databases that the server listens on:

Field	Description
fnm	The database name.
dfd,tfd1 tdfd2	The file descriptor.
lpa	The last page allocated.
lpw	The last page written (the database file size in pages).
nat	The number of active transactions.
use	The number of connected cursors.
flg	Flags.
lfd	The log file descriptor.
clf	The current log file number.
lfs	The log file size in kilobytes.
cti	The checkpoint time interval in minutes.
flg	Flags.
nlb	The next log to backup.

Field	Description
nsl	The next log to backup (after a BACKUP SNAPSHOT).
spn	The transaction span limit (0 means no limit).
cpp, cpl,ftp, fap fbl, lbl	Log pointers.

The last part of the Databases section is a dump of the lock table. For example:

```
1 S004:006:006
```

The ‘1’ indicates the transaction number that owns the lock entry. In some cases, this is the process number for special locks.

The ‘S’ means that it is a shared lock. This can also be:

- X = Exclusive
- U = Update
- I = Increment

When a T is prefixed to S, U, or X, it signifies a temporary lock.

The ‘004’ is the group to which the lock belongs. SQLBase divides pages in a database into groups.

The first ‘006’ is the starting page number and the second ‘006’ is the ending page number.

Configuration

There are sections in the configuration file (sql.ini) specific to the graphical interface of the Windows 95 and Windows NT servers. This section is dbntsrv.gui.

In these sections, you can configure any or all of the following keywords:

dbwin	*statwin*
displevel	*syswin*
mainwin	*procwin*

Use the Save Settings and Clear Settings options of the server’s **File** menu to configure these keywords. Read *Chapter 3, Configuration (sql.ini)* for detailed information about the keywords.

Chapter 6

DBA Operations

This chapter steps you through some of the tasks you will perform as a DBA. It also indicates the methods available to perform the task. This chapter includes the following tasks:

- Starting and stopping SQLBase
- Creating and deleting a database
- Deinstalling and re-installing a database
- Loading and unloading a database
- Creating a read-only database
- Upgrading your SQLBase software
- Partitioning a database
- Maintaining a partitioned database
- Reorganizing a database
- Checking database integrity

Note: If you are a Windows 95 or Windows NT user, you can also use Centura's SQLConsole to perform most DBA operations listed in this chapter. From a single Windows desktop, SQLConsole provides an easy-to-use graphical interface that lets you perform DBA operations, like database load and unload operations without the need for SQL commands. Read the *SQLConsole Guide* for details.

Starting and stopping SQLBase

This section shows you how to start and stop each of the SQLBase single-user database servers and multi-user database servers.

You must start a server before you can access a database from a client application.

Make sure that the files listed below are in a directory where SQLBase can find them. The best way to ensure this is to keep them in the same directory as the rest of the SQLBase software (C:\Centura, by default).

- error.sql
- message.sql
- sql.ini
- country.sql
- main.ini

NetWare

This section describes how to start and stop the SQLBase Server for NetWare, both the 3.x and 4.x versions. You must start the database server before clients can access a database.

Starting

SQLBase Server for NetWare is a NetWare Loadable Module (NLM) that runs on Novell's NetWare server operating system. An NLM is a program that you can load into or unload from server memory while the server is running. When loaded, an NLM is part of the NetWare operating system. When unloaded, an NLM releases the memory and resources that were allocated for it.

Loading the SQLBase Server for NetWare

1. Ensure that the *clib.nlm* and *mathlib.nlm* files are loaded. These files are Novell NLMs that contain C functions called by the SQLBase server.

   ```
   load clib
   load mathlib
   ```

   You can either load these manually from the command line, or edit the NetWare server’s *autoexec.ncf* file to include them. The *autoexec.ncf* file is a batch file that executes after the NetWare operating system boots. This file names the server and assigns it an internal network number, specifies settings for network boards, and loads LAN drivers.

2. Load *dfs.nlm*, which provides support for Novell's direct file system, or load *dfd.nlm*.

- If you are loading a SQLBase Server for NetWare 3.x, use the following commands:

  ```
  load directfs
  load dfs
  ```

  or

  ```
  load dfd
  ```

- If you are loading a SQLBase Server for NetWare 4.x, use the following command:

  ```
  load dfd
  ```

  or

  ```
  load dfs
  ```

Note: You do not need to explicitly load *directfs.nlm* on NetWare 4.x.

3. Load the SQLBase support programs and server program in this order. This example loads the 3.x multi-user server:

   ```
   load nlm:\centura\dll
   load nlm:\centura\spxdll
   load nlm:\centura\dbnservr
   ```

Note: For TLI-TCP/IP support, replace the spxdll file above with *tlidll*.

This example loads the 4.x multi-user server:

```
load nlm:\centura\dll
load nlm:\centura\spxdll40
load nlm:\centura\dbnwsrv
```

Note: For TLI-TCP/IP support, replace the spxdll40 file above with tlidll.

If you want to start the SQLBase server every time the NetWare server boots, add lines to the *autoexec.ncf* file that load *dfs.nlm*, *dll.nlm*, *spxdll40.nlm*, or *spxdll.nlm*, and the database server.

If you do not want to start the SQLBase server every time the NetWare server boots, you can create a *sqlbase.ncf* file that loads *dfs.nlm*, *dll.nlm*, *spxdll40.nlm*, or ***spxdll.nlm***, and the database server.

The database server is removed from memory when the computer is switched off or rebooted. You must start the database server after each startup or reboot.

Stopping

You should stop the SQLBase server before turning the database server computer off. To stop the SQLBase server:

1. Press the **Esc** key.
2. If there are users connected to the server, a dialog box appears asking you to confirm that you want to stop the server. Press **Y** to bring the server down gracefully.

After stopping SQLBase, the server unloads itself. You can unload the support NLMs to free resources. This example unloads the 3.x NLMs:

```
unload spxdll
unload dll
unload dfs
unload directfs
```

This example unloads the 4.x NLMs:

```
unload spxdll40
unload dll
unload dfs
```

Be sure to unload the same files you loaded. For example, if you loaded the TLI-TCP/IP file *tlidll* instead of *spxdll* or *spxdll40*, specify *tlidll* in the UNLOAD command.

Note: You cannot unload the SQLBase NLM from the console prompt of the NetWare server; you must always use the ESC key to bring down the server. Never attempt to unload the SQLBase NLM before gracefully shutting it down. Once loaded, the SQLBase NLM becomes a NetWare resource and must be closed before it can be unloaded. NetWare can crash if you unload SQLBase *before* pressing the ESC key.

Windows

This section describes how to start and stop the SQLBase Server for Windows 3.x.

Multi-tasking server

Starting

To start *dbwservr.exe*, first specify it with SQLEdit. This corresponds to a *comdll=sqlwsv* entry in the [winclient.dll] section of the server’s sql.ini file.

You can start the server by itself by double-clicking on its icon, but this is not required before starting a Windows client application. When you attempt to access a database via a Windows application, *dbwservr.exe* is loaded into Windows memory automatically.

Stopping

To stop the *dbwservr.exe* server, click the DBWServr icon and choose the **Close** option.

Windows 95 and Windows NT

This section describes how to start and stop the SQLBase Server for Win32 environments. You must start the database server before clients can access a database.

Both the Windows 95 and Windows NT server use the Windows Program Manager interface. For information on this interface, see the *GUI Interfaces* section of the *DBA Interfaces* chapter.

Starting

To start either server, do the following:

- If an icon exists for the server program, double click on it.
- Otherwise, select **File, Run** or **Start, Run** and enter **dbntsrv**. For example, the following command starts the SQLBase Server for Windows NT that was installed in the \Centura directory:

  ```
  c:\centura\dbntsrv
  ```

  Click **OK**.

The server is removed from memory when the computer is switched off or rebooted. You must start the database server after each startup or reboot.

Stopping

You should stop the SQLBase server before turning the database server computer off. To do so, select **Exit** from the **File** menu of the SQLBase menu bar.

Setting configuration keywords

You can set the value of some configuration keywords (such as *dbdir*) when you start a server program. Specify a keyword and its value on the command line to override a sql.ini configuration entry.

For example, the following command starts SQLBase Server for NT and specifies the database home directory (*dbdir*) as *d:\test\db*. This overrides a *dbdir* configuration entry in the [dbntsrv] section of the sql.ini file.

```
dbntsrv dbdir=d:\test\db
```

You can specify more than one keyword on the command line, each separated by a space, for example:

```
dbntsrv dbdir=d:\test\db log=d:\test\dblog
```

Read *Chapter 3, Configuration (sql.ini)* for more for detailed information on all the configurable keywords.

Creating a database

You must establish a server connection to create a database. You can do this with SQLTalk's SET SERVER command or the SQL/API's *sqlcsv* function.

There are a number of ways to create a database. Whatever interface you use, the result is the same: the database template file, *start.dbs*, is copied to a new *<database_name>.dbs* file.

SQLTalk	Use the CREATE DATABASE command to create a database and install it on the network.
SQL/API	Use the *sqlcre* function to create a database.
From an operating system prompt	Use an operating system command to manually copy *start.dbs* to a new file.

This example shows how to create a database using SQLTalk:

```
SET SERVER SERVERP/PASSWORD;
SHOW DATABASES ON SERVER SERVERP;
*****
DATABASES ON SERVER: SERVERP
  DEMO
*****
CREATE DATABASE EMP;
DATABASE CREATED
SHOW DATABASES ON SERVER SERVERP;
*****
DATABASES ON SERVER: SERVERP
  DEMO
  EMP
*****
```

Deleting a database

You must establish a server connection to delete a database. You can do this with SQLTalk's SET SERVER command or the SQL/API's *sqlcsv* function. There are a number of ways to delete a database.

SQLTalk	Use the DROP DATABASE command to delete a database and its log files. SQLBase deletes both the database and log file directories (if they are different). This command also removes the *dbname* keyword from the server's configuration file (sql.ini).
SQL/API	Use the *sqldel* function to delete a database.
From an operating system prompt	Use an operating system delete command to manually delete a *.dbs* file.

This example shows how to delete a database in SQLTalk:

```
SET SERVER SERVERP/PASSWORD;
SHOW DATABASES ON SERVER SERVERP;
*****
DATABASES ON SERVER: SERVERP
  DEMO
  EMP
*****

SHOW CONNECT;
*****
Username: DBA
Database: DEMO
Cursor number:  1
*****

DROP DATABASE EMP;
SHOW DATABASES ON SERVER SERVERP;
*****
DATABASES ON SERVER: SERVERP
  DEMO
*****
```

Deinstalling a database

You must establish a server connection to deinstall a database. You can do this with SQLTalk's SET SERVER command or the SQL/API's *sqlcsv* function. You can take a database offline to prevent users from logging on to it. This is useful when you want to restore and recover a database. There are a number of ways to deinstall a database.

SQLTalk	Use the DEINSTALL DATABASE command to remove a database from the network. This command also removes the *dbname* configuration entry from the server's configuration file (sql.ini).
SQL/API	Use the *sqlded* function to deinstall a database. This command also removes the *dbname* configuration entry from the server's configuration file (sql.ini).
From an operating system prompt	Via an editor, remove the *dbname* configuration entry from the appropriate section of the server's configuration file (sql.ini).

This example shows how to deinstall a database using SQLTalk:

```
SET SERVER SERVERP/PASSWORD;
SHOW DATABASES ON SERVER SERVERP;
*****
DATABASES ON SERVER: SERVERP
  DEMO
  EMP
*****

SHOW CONNECT;
*****
Username: DBA
Database: DEMO
Cursor number:  1
*****

DEINSTALL DATABASE EMP;
SHOW DATABASES ON SERVER SERVERP;
*****
DATABASES ON SERVER: SERVERP
  DEMO
*****
```

Re-installing a database

You must establish a server connection to re-install a database. You can do this with SQLTalk's SET SERVER command or the SQL/API's *sqlcsv* function. There are a number of ways to re-install a database.

SQLTalk	Use the INSTALL DATABASE command to re-install a database on the network. This command also adds a *dbname* entry to the server's configuration file (sql.ini).
SQL/API	Use the *sqlind* function to re-install a database. This command also adds a *dbname* entry to the server's configuration file (sql.ini).
From an operating system prompt	Via an editor, add a *dbname* configuration entry to the appropriate section of the server's configuration file (sql.ini).

This example shows how to re-install a database using SQLTalk:

```
SET SERVER SERVERP/PASSWORD;
SHOW DATABASES ON SERVER SERVERP;
*****
DATABASES ON SERVER: SERVERP
  DEMO
*****

SHOW CONNECT;
*****
Username: DBA
Database: DEMO
Cursor number:  1
*****

INSTALL DATABASE EMP;
SHOW DATABASES ON SERVER SERVERP;
*****
DATABASES ON SERVER: SERVERP
  DEMO
  EMP
```

Loading and unloading a database

Unloading data involves “dumping” database information and Data Definition Language (DDL) commands to a flat file. You can then load this information to a new database. You can also use the load operation to load information to SQLBase from an external flat file.

SQLBase uses the server to perform load and unload processes. This relieves the client of unnecessary processing, reduces network traffic, and allows the server to load and unload the data locally on its own machine. It also means that you can use the SQL/API *sqlrlo* and *sqlwlo* functions to read and write data, a buffer at a time.

Within a stored command or procedure, you can load and unload to the server but *not* to the client.

SQLBase load and unload operations handle large amounts of data, especially for large tables or databases. Load/unload files consume storage space and can be expensive to transmit. There is a compression option to reduce the size of these external files. You can compress the external file when you unload it, and decompress it when you reload it. Compression occurs at the server before transmission over the network.

Loading

The following are methods to load database information:

SQLTalk	Use the LOAD command to load database information from a flat file.
SQL/API	Use the *sqlldb* function to load database information from a flat file.

A load operation can restore data from a backup file created with a SQLBase unload operation, or can enter data into the database from an external file. The external file can be in SQL, ASCII or DIF format. You can create the file either manually or with the SQLBase unload methods listed in the next section. Do not edit these files manually. If you try to add commands such as COMMIT or ROLLBACK to the file, the load will fail.

Note that during the load operation, if ALTER TRIGGER commands are encountered, they are automatically processed.

Unloading

The following are methods to unload database information to a flat file:

SQLTalk	Use the UNLOAD command to unload database information to a flat file.
SQL/API	Use the *sqlunl* function to unload database information to a flat file.

A SQLBase unload operation allows you to back up a database or transfer data from a database to another program through interchange formats. An unload does not unload invalid stored commands.

When triggers are encountered during an unload operation, the status of the trigger is checked. If the trigger is disabled, the unload operation generates an ALTER TRIGGER *triggername* DISABLE statement for that trigger, which immediately follows the trigger's CREATE TRIGGER statement. If the trigger is enabled, the unload operation takes no action.

Segmenting a LOAD/UNLOAD file

You can split a load/unload external file into segments spanning multiple disks. Segmenting an external file allows you to unload and load databases that might exceed single disk or system unit limits. For example, since there is a 2 gigabyte size limit imposed for non-partitioned databases, you may need to unload a database that exceeds 2 gigabytes into multiple file segments. Segmenting external files also lets you take advantage of available space that is spread out over several disks.

When unloading to a segmented external file, you designate how many bytes of information should reside in each file segment, and where each segment will reside. SQLBase unloads the database information sequentially to the file segments, putting the specified amount of information in each file segment. You can create a segmented file for any type of UNLOAD.

Control file

To manage the file segments, SQLBase uses a *control file*. This file contains the following information:

- A prefix for the file segment names
- The destination directories where the file segments reside
- The maximum size of each file segment (for an unload operation only)

There are two types of control files for segmenting a LOAD/UNLOAD file:

- unload

- load

Unload control file. You must create the unload control file yourself using the instructions in the following section, Creating the control file. When running the unload operation, you then specify this control file name with the CONTROL clause. SQLBase automatically generates and unloads the database information sequentially to the file segments named in the control file.

Create the control file at the same location where you are going to store the external files. You can create the unload control file on either the client or server machines. When you run UNLOAD, use the ON SERVER or ON CLIENT clause to tell SQLBase where the file is located. When it unloads information, SQLBase generate the unload file segments on the same machine as the unload control file, including any network drives.

If you do not supply a specific path when you run the unload operation, SQLBase assumes that the unload control file resides in the \Centura directory on the machine (or connected network drive) as specified by the ON SERVER or ON CLIENT clause.

Load control file. When you run an unload command with a control file, SQLBase automatically generates a corresponding *load control file* as output, as well as the unload file segments. When you subsequently run the load operation, use this load control file name.

You should use the SQLBase-generated load control file whenever possible, without making changes to it. However, certain situations may require a new or modified load control file, such as loading information that was unloaded from a non-SQLBase database, or moving the file segments since the UNLOAD. To create or modify a load control file, use the guidelines shown in the next section, Creating the control file.

SQLBase puts the new load control file in the same directory as the unload control file you created. If you do not supply a specific path, SQLBase assumes that the load control file resides in the \Centura directory.

The file segments must either reside on or be accessible from the same machine as the load control file.

Creating the control file

To create a control file, use any editor on your system. The file must be in ASCII format, and contain the following information:

```
FILEPREFIX  <filename prefix>
DIR   <destination dir>SIZE  <maximum size of the unload
    segment file in megabytes>
DIR   <destination dir>SIZE  <maximum size of the unload
    segment file in megabytes>
...
```

Note: The SIZE parameter only appears in the unload control file, not the load.

The following subsections describe these fields.

- FILEPREFIX

  This parameter names the unload file segments. SQLBase names each unload file segment with this prefix, and appends an ascending value *n*. It also uses this file prefix to name the load control file, appending an extension of *.lcf*.

  Example:

  If you specify a file prefix *dbs*, SQLBase generates the following file segments during the UNLOAD operation:

  ```
  dbs.1
  dbs.2
  dbs.3
  ...
  ```

  SQLBase also creates a load control file called *dbs.lcf*

  The number of segments you can create is limited only by your system; SQLBase itself does not set a limit.

Note: Make certain that there are no existing files in the unload control file directory with the same name as the eventual load control file. SQLBase will overwrite any existing files that have the same name as the load control file.

- DIR

  Use this parameter to name the destination directory (Windows Servers) or volume name (NetWare Server) for the file segments. You can put all the file segments in one directory, or split them out to different directories. SQLBase unloads the information to the file segments according to their listed order in the control file.

  Each line specifies one file segment.

  The directories can also include network drives, as long as the destination machine can access these drives.

- SIZE

  This parameter specifies the maximum integer size (in megabytes) allowed for the specified file segment. During an unload operation, SQLBase dumps this exact amount of data to that file segment.

  This parameter only appears in the unload control file, not the load.

Valid values are null or integer values of 1 through 2048 (in megabytes). Use the following calculation to determine the maximum number of bytes you can allocate:

bytes = (size * 1048576)

which is the maximum file size common across most systems.

If you wish to limit the size of your files, you can specify multiple file segments of a certain size, like the following Windows NT example:

Example:

```
DIR c:\unldir\ SIZE 50
DIR c:\unldir\ SIZE 50
DIR c:\unldir\ SIZE 50
```

With this control file, SQLBase unloads 50 megabytes of information to each of the three file segments.

The sum of all the sizes must be large enough to write all the unload information. If you do not know how much space you need, you can specify an approximate number of segments and their size. Depending on the amount of data you have, the last file segment SQLBase dumps to may or may not reach the SIZE amount. Any leftover segments are ignored; they do not tie up the space you designated for them.

Note: External files are not pre-allocated. If you direct SQLBase to create external files on the server machine, remember to consider the growth of log, history, and temporary files when you estimate the necessary available free space.

Note that since SQLBase dumps exactly the amount of data you specify, database objects and associated data may be split across multiple files.

If you build your own load control file, SQLBase ignores any SIZE parameters you create.

Control file example

The following Windows NT example shows an unload control file:

```
FILEPREFIX  dbs
DIR  c:\unldir\ SIZE  100
DIR  d:\unldir\ SIZE  50
DIR  e:\unldir\ SIZE  200
```

Note that while SQLBase ignores any white space in this file, you cannot “wrap” a command statement to continue on the next line; SQLBase interprets new line characters as the end of an entry.

During an unload operation, this control file tells SQLBase to unload database information in the following order:

1. 100 megabytes of information to a file called **c:\unldir\dbs.1**
2. 50 megabytes of information to a file called **d:\unldir\dbs.2**
3. 200 megabytes to a file called **e:\unldir\dbs.3**

SQLBase also generates a corresponding load control file called *dbs.lcf* in the same directory as the unload control file. The following Windows NT example shows this file:

```
FILEPREFIX  dbs
DIR  c:\unldir\
DIR  d:\unldir\
DIR  e:\unldir\
```

SQLBase loads the information in the order listed. Note that there is no SIZE parameter.

Error logging and recovery

Both the SQLBase loading and unloading commands can create a message log file. These message log files are useful for tracking activities and errors occurring during the load and unload processes.

For SQL or ASCII (not DIF) format, the LOAD command also has a option to restart a load operation from a particular line number in the load file. If you fix a data or statement error encountered during a load operation, use this clause to restart the load at the point where the error occurred, rather than having to restart the load from the beginning.

The line number for the START AT clause for a DDL statement must be the first line of the DDL command. For an INSERT, the line number for the START AT clause must be either the first line of the INSERT command or one of the line numbers that corresponds to a row of data you are inserting.

Note that with segmented loads, the line numbers are cumulative in the load file segments. To restart a segmented load from a specific line number, you must determine yourself in which file segment the specified line number is located.

Note: To avoid possible loss of data, issue a COMMIT statement immediately after a data or statement load error unless you are running SQLTalk in BAT mode, which performs an implicit COMMIT upon exiting.

Network or server failures are generally not recoverable. The existing loaded data is rolled back; you cannot restart the load from a specific line in this situation.

Load performance enhancements

To improve load performance and preserve space, issue the following commands when executing a load:

```
set recovery off;
lock database;
load [sql | ascii | diff] external_file on server;
commit;
unlock database;
set recovery on;
```

Creating a read-only database

SQLBase allows you to create read-only databases. This is useful for CD-ROM drives, as well as for other purposes:

- You can ensure that a database is altered only during certain hours of the day.
- You can have one central updatable database and distribute read-only copies of it to other sites.

To establish a database as read-only, use SQLTalk's SET command:

```
SET READONLYDATABASE ON;
```

You can turn READONLYDATABASE on or off from a C program using the *sqlset* function with the SQLPROD parameter.

You must be the only user connected to the database to turn READONLYDATABASE on or off. The default setting is off.

To display the setting of READONLYDATABASE, use SQLTalk's SHOW command:

```
SHOW READONLYDATABASE;
```

Read-Only isolation level is disabled when READONLYDATABASE is on. You cannot set the isolation level of your transaction to Read-Only mode.

Transaction log files are disabled once READONLYDATABASE has been turned on.

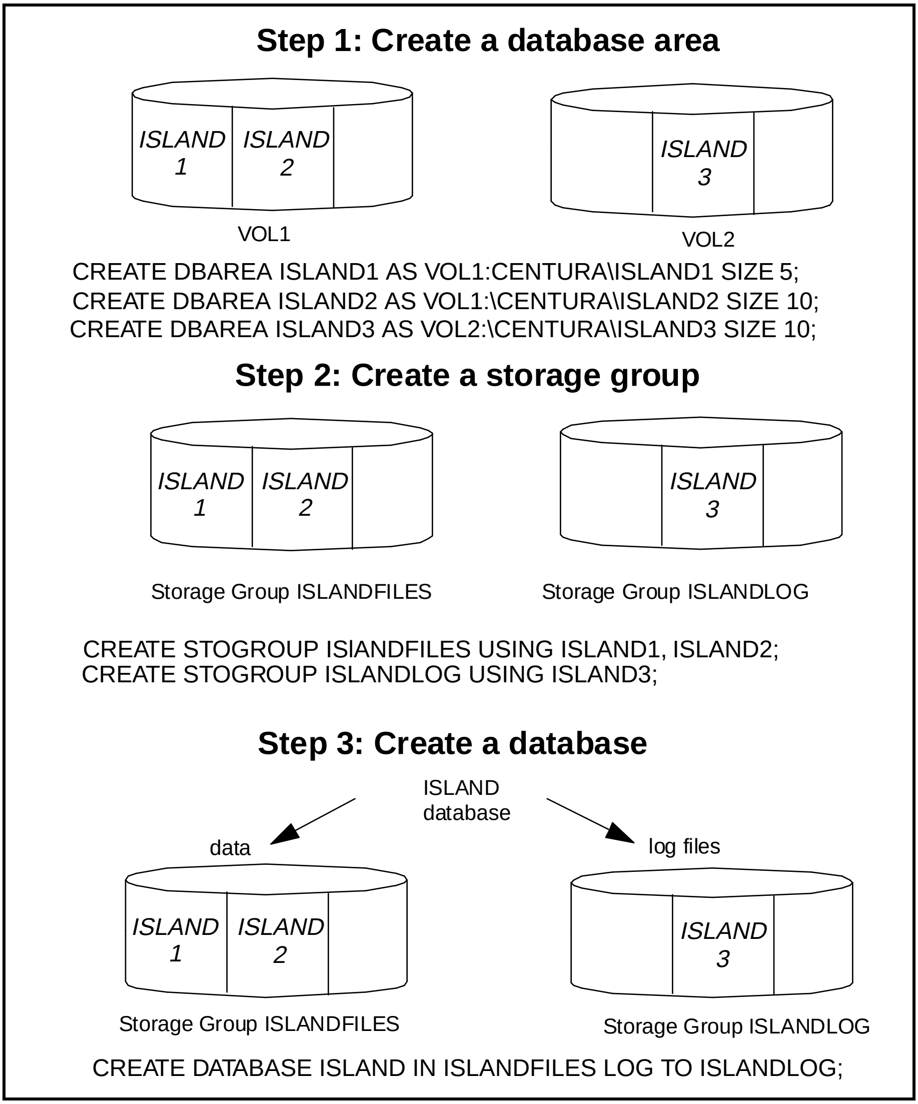
To turn off read-only mode, enter the following command:

```
SET READONLYDATABASE OFF;
```

You must also set the configuration file's *tempdir* keyword or your operating system's file *tempdir* environment variable to tell SQLBase where to store temporary files. SQLBase actually creates a subdirectory in the directory pointed to by *tempdir* with the same name as the database. It is here that it stores the temporary files.

Partitioning a database

This section covers setting up a database on partitioned volumes. It also introduces commands to maintain a partitioned database.



```
CREATE DBAREA ISLAND1 AS VOL1:CENTURA\ISLAND1 SIZE 5;
CREATE DBAREA ISLAND2 AS VOL1:\CENTURA\ISLAND2 SIZE 10;
CREATE DBAREA ISLAND3 AS VOL2:\CENTURA\ISLAND3 SIZE 10;
```

```
CREATE STOGROUP ISlANDFILES USING ISLAND1, ISLAND2;
CREATE STOGROUP ISLANDLOG USING ISLAND3;
```

```
CREATE DATABASE ISLAND IN ISLANDFILES LOG TO ISLANDLOG;
```

Steps for partitioning a database

Step 1: creating a database area

The first step in partitioning a database is to create a database area of a specified size either within the file system or in a raw partition. This process allocates room in which to create storage groups for your databases.

For example, to create the database area ISLAND1 in the \Centura directory in volume VOL1 with a size of 5 megabytes, specify:

```
CREATE DBAREA ISLAND1 AS VOL1:\CENTURA\ISLAND1 SIZE 5;
```

Specify a name for the database area with no more than 18 characters and specify the size of a database area in megabytes. Specifying a database area size is mandatory. The maximum size is limited by available disk space.

SQLBase returns an error if the file already exists or the disk space is already being used for another database area. It also returns an error if a file of the specified size cannot be created or if the actual size of the partition is smaller than the specified size of the area.

Step 2: creating a storage group

The next step is to create a storage group (STOGROUP) by giving a name to a list of database areas. For example, to create the storage group ISLANDFILES for database areas ISLAND1 and ISLAND2, specify:

```
CREATE STOGROUP ISLANDFILES USING ISLAND1, ISLAND2;
```

Specify a name for the storage group with no more than 18 characters.

In the Novell environment, if the NetWare volumes containing the database areas are not mounted, SQLBase returns an error when you try to create a database.

Step 3: creating a database

The last step is to create a new database in a storage group. You can also select a separate storage group for the log files.

For example, to create the database ISLAND in the storage group ISLANDFILES, and place the log files in a separate storage group called ISLANDLOGS, specify:

```
CREATE DATABASE ISLAND IN ISLANDFILES LOG TO ISLANDLOG;
```

Specify a name for the database with no more than eight characters.

If you specify a database storage group but do not specify a storage group for the logs, log file space is allocated in the database storage group. If you do not specify a storage group at all, the default storage group in the MAIN database is used, if one exists. If the MAIN database does not exist or you do not specify a default storage group, SQLBase creates the database and places it in the normal file system.

A database and its logs should not share any disks.

You should spread a database across as many disks as possible.

Maintaining a partitioned database

This section steps you through tasks related to database areas and storage groups as well as extents.

Database areas

Changing the size of a database area

You can increase or decrease the size of a database area.

For example, to increase the size of the database area ISLAND1 to 6 megabytes, specify:

```
ALTER DBAREA ISLAND1 SIZE 6;
```

You can decrease the size of a database area only if the space being deleted is not in use. SQLBase returns an error message if the file cannot be re-sized to the specified size.

Adding or dropping an area from a storage group

You can add or drop a database area to or from a storage group.

For example, to add the database area ISLAND3 to the storage group ISlANDFILES, specify:

```
ALTER STOGROUP ISLANDFILES ADD ISLAND3;
```

To then drop the database area ISLAND3 from the storage group ISLANDFILES, specify:

```
ALTER STOGROUP ISLANDFILES DROP ISLAND3;
```

These changes do not affect the storage of existing databases in the storage groups, but they do affect the future allocation of space for databases or log files which use the STOGROUP.

Deleting a database area

You can delete a database area as long as there are no database or log files using space within the area, and/or it is being used by a storage group.

For example, to delete the database area ISLAND1, specify:

```
DROP DBAREA ISLAND1;
```

Storage groups

Setting a default storage group

Once you create a storage group, all subsequent attempts to create a database result in SQLBase creating a partitioned database.

You can designate one of the existing storage groups as the default storage group.

For example, to set the default storage group to ISLANDFILES, specify:

```
SET DEFAULT STOGROUP ISLANDFILES;
```

The above example assumes that you already created the database area ISlAND1 and the storage group ISLANDFILES.

If you omit the storage group name, the default is set back to null. This signals SQLBase to create databases in the normal (non-partitioned) file system.

Changing the database storage group

You can change the storage group for a database or its logs. For example, to change the storage group for ISLAND to ISLANDLISTER, specify:

```
ALTER DATABASE ISLAND STOGROUP ISLANDLISTER;
```

Existing database or log file space is not moved or affected by this command. The database you alter must already exist, or SQLBase returns an error message.

Deleting a storage group

You can also delete a storage group if it is not in use by any database and it is not currently the default storage group.

For example, to delete the storage group ISLANDFILES, specify:

```
DROP STOGROUP ISLANDFILES;
```

Extents

Setting the extension size

SQLBase databases grow dynamically as data is added and expand in units called extents. When a database file becomes full, SQLBase must add another extent to the database. You are responsible for determining the size of the extent.

SQLTalk’s SET EXTENSION command sets the extension size for both partitioned and non-partitioned databases. This is the syntax of the command:

```
SET EXTENSION <#kbytes>;
```

For partitioned databases, the value is rounded up to a 1 megabyte multiple.

Free space in partitioned databases

To determine how much free space is left in a partitioned database, use the following query:

```
SET LINEWRAP ON;
column 1 width 8 heading ‘Database’;
column 2 width 8 heading ‘stoareas’;
column 3 width 8 heading ‘areaname’;
column 4 width 8 heading ‘pathname’;
column 5 width 8 heading ‘areasize’;
column 6 width 8 heading ‘free%’;
break on 1 2;
select a.name, b.stogroup, b.areaname, @trim(c.pathname),
   c.areasize, @nullvalue(100*d.extsize/c.areasize),0)
from sysql.databases a,
   syssql.stoareas b,
   syssql.areas c,
   syssql.freeexts d
where
   b.arename = c.name
and c.name = d.name (+)
and ((a.stogroup = b.stogroup)
or  (a.logstogroup=b.stogroup))
order by 1,2
```

Reorganizing a database

There are two types of fragmentation that can affect performance:

- Fragmentation of the database (**.dbs*) file.

  This happens when the database file does not have contiguous disk space.

- Fragmentation of tables within a database file.

  The tables in a database file can become fragmented over time by modifications to the data.

You can use a disk fragmentation utility such as Norton Utilities’ Speed Disk (*sd*) to defragment a database file. Be sure to back up your database file first and check the database after you run the utility.

You can use SQLTalk’s REORGANIZE command to defragment tables in the database to which you are currently connected. The REORGANIZE command:

- UNLOADs the database into a file
- Initializes the database
- Re-LOADs the database

SQLTalk uses a temporary file called *sqltmp.nnn* for unloading.

Note: Make a copy of the database file *before* executing this command. If an error occurs during the reorganization, both the temporary file and the database itself could be lost.

If you want to perform a complete defragmentation of both the database file and the tables in the database, first execute *sd* or a similar utility, and then execute SQLTalk's REORGANIZE command. For example:

```
REORGANIZE;
UNLOAD COMPLETED
INITIALIZING DATABASE
STARTING TO LOAD

CREATE TABLE DBA.EMP (I INTEGER)
PCTFREE 10
TABLE CREATED

INSERT INTO DBA.EMP VALUES(:1)
LOADING TABLE DBA.EMP

PROCESSING DATA
50 ROWS LOADED
50 ROWS INSERTED
LOAD COMPLETED
```

Checking database integrity

The SQL CHECK command performs integrity checks on an entire database or only specified portions of a database. Integrity checking consists of:

- Checking the integrity of the system and group free space data structures.
- Checking the system data and allocation structures.
- Verifying the table row count against the actual number of rows.
- Cross-checking each index against its base table.
- Checking the integrity of each row and index page.
- Ensuring that each page is part of an allocated structure or is on a free page list.

SQLBase reads every page of the database during an integrity check, resulting in the placement of a shared lock on each page.

Command	Description
CHECK DATABASE	This command performs an integrity check on the entire database.
CHECK DATABASE SYSTEM ONLY	This command verifies only system-defined tables and indexes; it ignores user-created tables and indexes.
CHECK INDEX	This command performs an integrity check on only the specified index.
CHECK TABLE	This command performs an integrity check on only the specified table and its indexes.
CHECK TABLE WITHOUT INDEXES	This command performs an integrity check on only the specified table and prevents SQLBase from verifying any indexes associated with the table.

If SQLBase finds an integrity violation, it stops the integrity check and reports an error to the user. If the problem occurred on a user-defined object, SQLBase identifies the name of the object. If the problem occurred on a system-defined object, SQLBase provides either a description of the object or the name of the object.

If SQLBase discovers an object with integrity problems, you should drop it. If SQLBase discovers more extensive integrity problems, you may decide to restore a backed-up copy of the database. For example:

```
CHECK DATABASE;
CHECK DATABASE SYSTEM ONLY;
CHECK INDEX idx1;
CHECK TABLE emp;
CHECK TABLE emp WITHOUT INDEXES;
```

Chapter 7

Security and Authorization

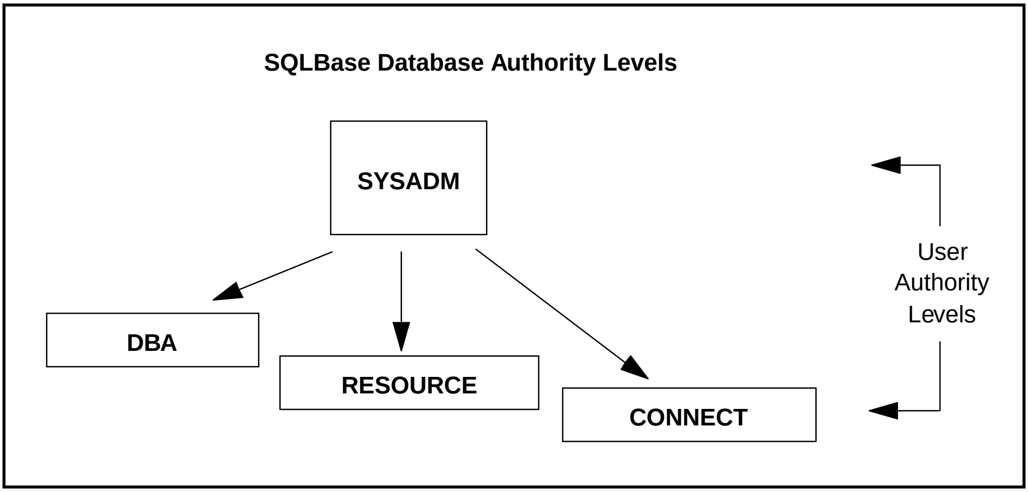
This chapter introduces you to SQLBase's security and authorization features.

- *Database authority* controls who can log on to SQLBase and the actions they can perform.
- *Table and view privileges* let users share some data while keeping other data private.
- *Server security* controls who can perform server administrative operations.
- EXECUTE privileges control who can access stored procedures.

Database authority

Database authority controls who can access a database and what they can do once connected to the database. SQLBase controls access with usernames and passwords.

SQLBase has 4 levels of users as shown in the following diagram.



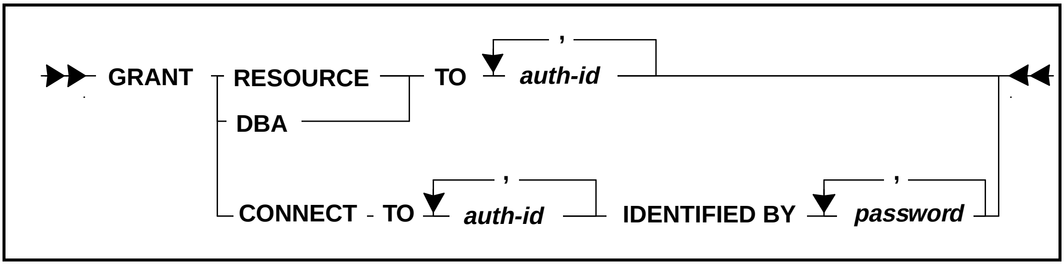
Database authority levels

The authority levels are hierarchical:

Authority Level	Description
SYSADM	Creates user accounts and designates users' authority levels and passwords.
DBA	Grants, changes, or revokes the object privileges of any user.
RESOURCE	Creates and drops objects. Grants, changes, or revokes the privileges of other users on those objects.
CONNECT	Accesses objects, but cannot create them.

Creating a user account

SYSADM alone has responsibility for establishing an account for each user by assigning each a username (auth-id), database authority level, and password. SYSADM does this using the GRANT command:



Only SYSADM can use this form of the GRANT command.

SYSADM cannot grant other users SYSADM authority nor change the SYSADM username. The only thing that can be changed for SYSADM is the password.

The following authority levels can be granted by SYSADM:

Authority Level	Description
CONNECT	This authority level must be granted before any other. This lets a user: • Logon to the database • SELECT from other users' tables and views if the SELECT privilege has been granted to the user or to PUBLIC • INSERT, UPDATE, and DELETE data into and from other users' tables if the appropriate privileges have been granted to the user or to PUBLIC • Create views and synonyms
RESOURCE	This authority level gives a user all CONNECT privileges and the right to: • Create tables and drop those same tables • GRANT, change, or REVOKE privileges for those tables to and from other users

Authority Level	Description
DBA	This authority level gives a user all CONNECT and RESOURCE privileges and all privileges on any object in the database as well as the right to: • GRANT, change, or REVOKE privileges for any object to and from other users

Read the *SQL Language Reference* for details about the GRANT command.

SQLTalk examples

The following example establishes an account with CONNECT database authority, the username *user1*, and the password *secret*:

```
GRANT CONNECT TO USER1 IDENTIFIED BY SECRET;
CONNECT AUTHORITY GRANTED
```

The example below increases *user1*'s authority level to RESOURCE:

```
GRANT RESOURCE TO USER1;
RESOURCE AUTHORITY GRANTED
```

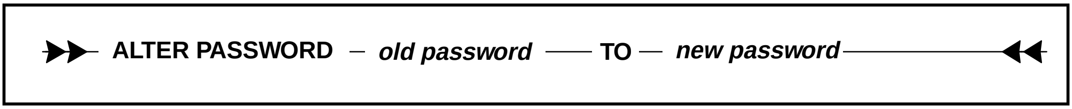
The example below increases *user1*'s authority level to DBA:

```
GRANT DBA TO USER1;
DBA AUTHORITY GRANTED
```

Changing a user’s password

There are two ways to change a user’s password:

- SYSADM can change a user's password using the GRANT command and specifying the CONNECT option.
- A user can change his own password using the ALTER PASSWORD command.



Read the ***SQL** Language Reference* for details about the ALTER PASSWORD command.

SQLTalk examples

The following example shows SYSADM changing *user1*'s password from *secret* to *erehwon*:

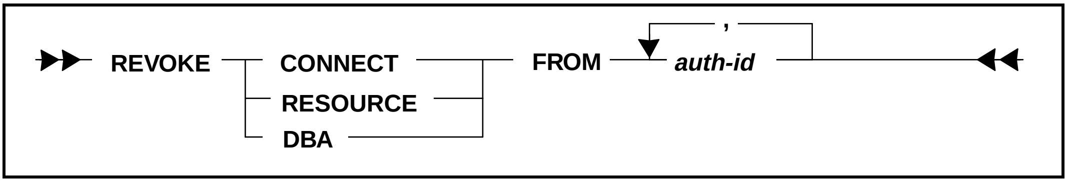
```
GRANT CONNECT TO USER1 IDENTIFIED BY EREHWON;
CONNECT AUTHORITY GRANTED
```

The example below shows *user1* changing his password from *secret* to *erehwon*:

```
ALTER PASSWORD SECRET TO EREHWON;
PASSWORD ALTERED
```

Revoking a user’s authority

The REVOKE command changes a user’s database authority level.



SYSADM can revoke these authority levels:

Authority Level	Description
DBA	This reduces a user’s database authority to the CONNECT level: the user can no longer create or drop tables, or grant or revoke privileges on those table from other users. Any and all tables and views previously created by the user are kept intact.
RESOURCE	This reduces a user’s database authority to the CONNECT level: the user can no longer create or drop tables, or grant or revoke privileges on those table from other users. Any and all tables and views previously created by the user are kept intact.
CONNECT	This disallows the user access to the database. SYSADM must revoke privileges on tables and views *before* revoking CONNECT authority.

Read the *SQL Language Reference* for details about the REVOKE command.

SQLTalk examples

The following example takes away *user1*'s DBA authority:

```
REVOKE DBA FROM USER1;
DBA AUTHORITY REVOKED
```

The example below removes *user1*'s CONNECT authority:

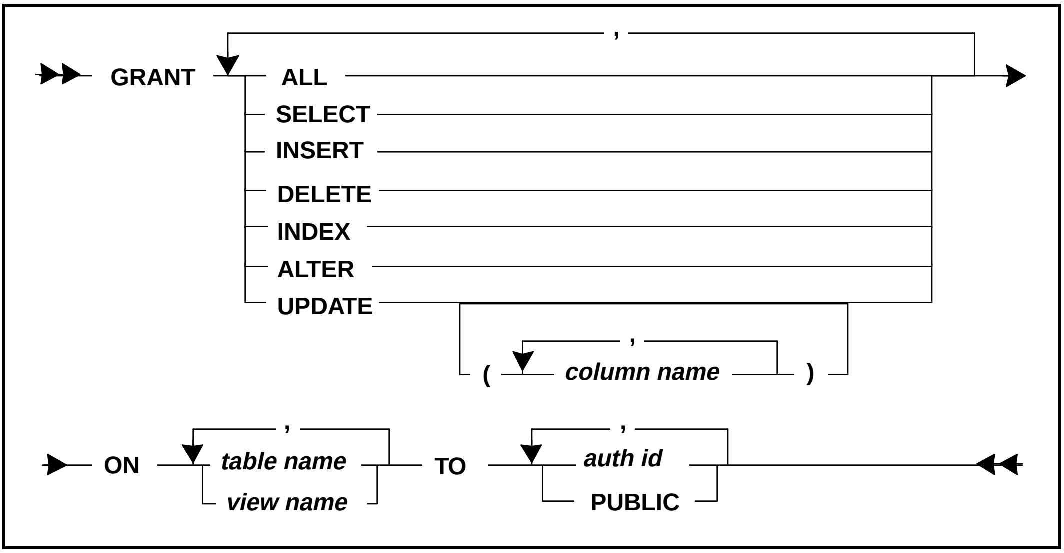
```
REVOKE CONNECT FROM USER1;
CONNECT AUTHORITY REVOKED
```

Table and view privileges

A user with DBA authority can grant privileges on *any* tables or views in the database. A user with RESOURCE authority can grant privileges on only those tables created by him or on views based completely on tables created by him. Because a user with only CONNECT authority cannot create objects, he cannot grant privileges either.

Granting privileges

A user who creates a table or view is the *owner* of that table or view and has all privileges on it. An owner can grant privileges on objects he owns to other users. He does this with the GRANT command.



GRANT INDEX and GRANT ALTER do not apply to views.

As an owner, you can grant the following privileges:

Privileges	Description
SELECT	Select data from a table or view.
INSERT	Insert rows into a table or view.
DELETE	Delete rows from a table or view.
UPDATE	Update a table and (optionally) update only the specified columns.
INDEX	Create or drop a table's indexes.
ALTER	Alter a table.
ALL	Exercise all of the above table or view privileges.

The keyword PUBLIC represents all users. By granting a privilege to PUBLIC, it means that all current and future users have the specified privilege on the table or view.

Read the *SQL Language Reference* for details about the REVOKE command.

SQLTalk examples

In the following example, *user1* gives *user2* the privilege to look at the data in *emp_info*:

```
GRANT SELECT ON EMP_INFO TO USER2;
PRIVILEGE(S) GRANTED ON TABLE OR VIEW
```

The example below shows a DBA granting *user2* the privilege to look at the data in *user1*'s *emp_info* table:

```
GRANT SELECT ON USER1.EMP_INFO TO USER2;
PRIVILEGE(S) GRANTED ON TABLE OR VIEW
```

The following example shows a DBA restricting *user2*'s UPDATE privilege on the *phoneno* column in *user1*'s *emp_info* table:

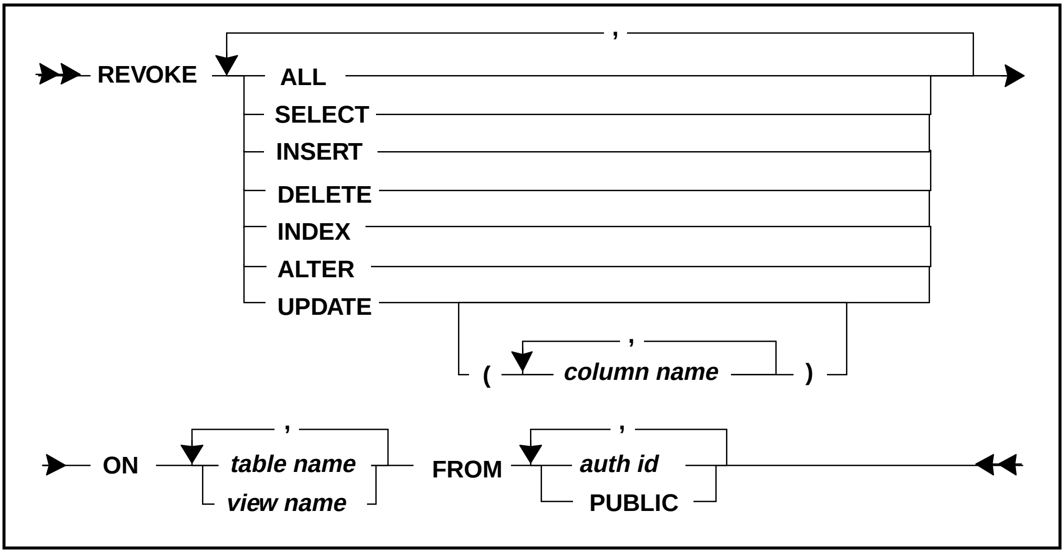
```
GRANT SELECT, UPDATE (PHONENO) ON USER1.EMP_INFO TO USER2;
PRIVILEGE(S) GRANTED ON TABLE OR VIEW
```

The next example shows a DBA giving all users the privilege of accessing the data in *user1.emp_info*:

```
GRANT SELECT ON USER1.EMP_INFO TO PUBLIC;
PRIVILEGE(S) GRANTED ON TABLE OR VIEW
```

Revoking privileges

The REVOKE command revokes privileges previously granted to a user.



Any user with the appropriate privileges for a table can revoke the privileges for the corresponding views.

The privileges are the same as for the GRANT command.

REVOKE INDEX and REVOKE ALTER do not apply to views.

The keyword PUBLIC represents all users. By revoking a privilege from PUBLIC, it means that all current users no longer have the specified privilege on the table or view.

Read the *SQL Language Reference* for details about the REVOKE command.

SQLTalk examples

The following example shows *user1* revoking *user2*'s privilege to access the data in *emp_info*:

```
REVOKE SELECT ON EMP_INFO FROM USER2;
PRIVILEGE(S) REVOKED ON TABLE
```

The example below shows a DBA revoking *user2*'s privilege to access the data in *user1.emp_info*:

```
REVOKE SELECT ON USER1.EMP_INFO FROM USER2;
```

```
PRIVILEGE(S) REVOKED ON TABLE
```

The following example shows a DBA revoking *user2*’s UPDATE privilege on the *phoneno* column.

```
REVOKE UPDATE (PHONENO) ON USER1.EMP_INFO FROM USER2;
PRIVILEGE(S) REVOKED ON TABLE
```

The example below shows a DBA revoking the public’s privilege to the data in *user1.emp_info*:

```
REVOKE SELECT ON USER1.EMP_INFO FROM PUBLIC;
PRIVILEGE(S) REVOKED ON TABLE
```

Synonyms

When you access a table, view, or external function created by another user (once you have been granted the privilege), you must fully-qualify the object name by prefixing it with the owner's name:

```
authorization-id.object-name
```

For example:

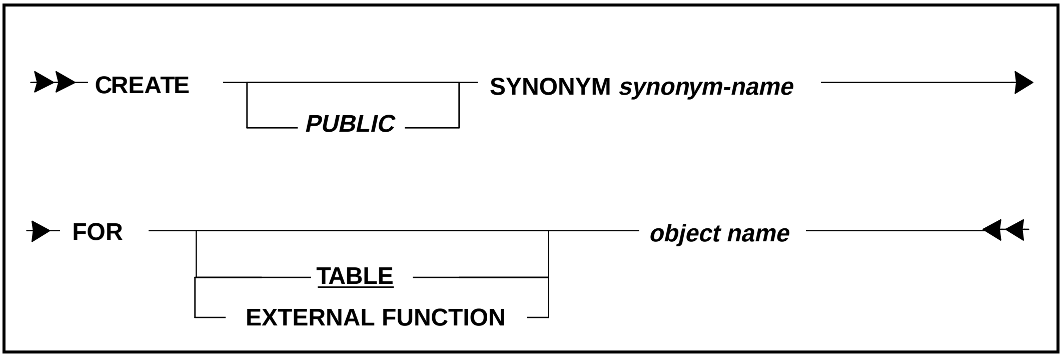
```
sysadm.systables
user1.emp_info
```

If you try to access another user’s table, view, or external function and you do not qualify the object name with the owner's name, SQLBase looks for a table, view, or external function owned by *you* with that name.

Synonyms save you typing by allowing you to refer to another user’s table, view, or external function without having to fully qualify the name. A synonym is another name for a table, view, or external function.

Creating a synonym

The CREATE SYNONYM command creates a synonym for the name of a table, view, or external function.



Synonyms for tables are stored in the SYSADM.SYSSYNONYMS system catalog table. Synonyms for external functions are stored in the SYSADM.SYSOBJSYN system catalog table.

You must have at least one privilege on the table, view, or external function to create a synonym.

If many users need to access a table, view, or external function, the SYSADM, a DBA, or the owner can specify the PUBLIC keyword in the CREATE SYNONYM command to create a public synonym for all users with privileges on the table. This saves each user from having to create his own synonym. Be sure that only one table, view, or external function with that name exists in the database.

Note that if you want to create an external function synonym, you must provide the keyword EXTERNAL FUNCTION in the FOR clause. If no object type is specified in the FOR clause, the object type defaults to TABLE.

Read the *SQL Language Reference* for details about the CREATE SYNONYM command.

SQLTalk examples

The following example shows a user creating a synonym for another user's (*user1*) table (*emp_info*):

```
CREATE SYNONYM EMP FOR USER1.EMP_INFO;
SYNONYM CREATED
```

The example below shows the DBA creating a public synonym for all the users:

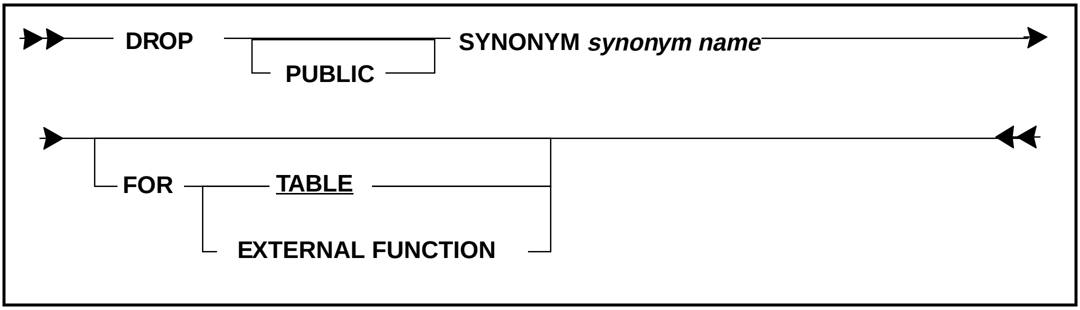
```
CREATE PUBLIC SYNONYM EMP FOR USER1.EMP_INFO;
SYNONYM CREATED
```

Note that in the previous examples, since no object type is included, the object type defaults to TABLE. These examples can also be presented as follows:

```
CREATE SYNONYM EMP FOR TABLE USER1.EMP_INFO;
CREATE PUBLIC SYSNONYM EMP FOR TABLE USER1.EMP_INFO;
```

Dropping a synonym

The DROP SYNONYM command deletes a specified synonym.



A synonym can only be dropped by its creator, SYSADM, or a DBA. SQLBase automatically drops any views based on the synonym as well.

Specify the PUBLIC keyword to remove a synonym that is available to all users.

Note that if you want to create an external function synonym, you must provide the keyword EXTERNAL FUNCTION in the FOR clause. If no object type is specified in the FOR clause, the object type defaults to TABLE.

Read the *SQL Language Reference* for details about the DROP SYNONYM command.

SQLTalk examples

The following example deletes a synonym:

```
DROP SYNONYM EMP;
SYNONYM DROPPED
```

The example below drops a PUBLIC synonym:

```
DROP PUBLIC SYNONYM EMP;
SYNONYM DROPPED
```

Note that in the previous examples, since no object type is included, the object type defaults to TABLE. These examples can also be presented as follows:

```
DROP SYNONYM EMP FOR TABLE;
DROP PUBLIC SYSNONYM EMP FOR TABLE;
```

Views

Granting privileges is one way of giving users access to database objects. Users can be granted select privileges without having privileges on the base table. The GRANT command gives you column level control by allowing you to specify whether users can see some or all of the columns in a table.

Views are another, more flexible way of giving users selective access to data. Views can enforce data security by allowing:

- Row-level access

  You can limit the rows that users can access by specifying a WHERE clause in the view definition.

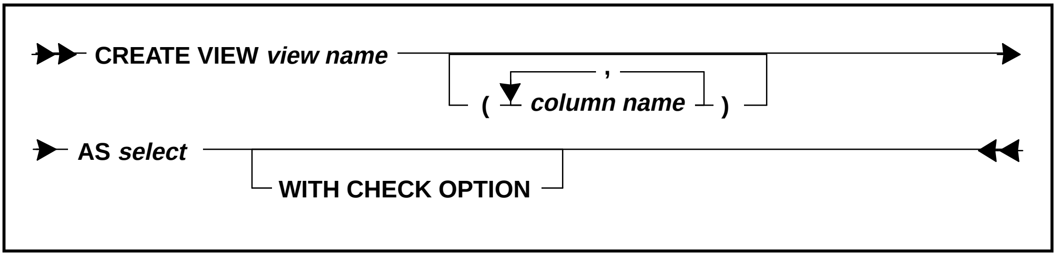
- Column-level access

  Like the GRANT command, you can limit the columns that users can access by specifying only some of the table's columns.

A user can create a view of a base table which only selects certain rows or columns and then grant privileges on that view.

Creating a view

The CREATE VIEW command creates a view on one or more tables or views.



The creator of a view needs to have SELECT privileges on the columns of the base table.

The WITH CHECK OPTION causes all inserts and updates through the view to be checked against the view definition and rejected if the inserted or updated row does not conform to the view definition. If the clause is omitted, then no checking happens.

Read *SQL Language Reference* for details about the CREATE VIEW command.

SQLTalk examples

The following examples use this table:

```
LNAME          DEPT      SALARY    PHONENO
========       =====     =======   =========
Smith          355       32654     832
Jones          150       28312     285
Brown          700       48233     721
Joyce          150       87935     145
Chen           355       37666     222
Herrlein       700       18150     185
```

A DBA creates the view below so that *user1* can view the salaries and telephone numbers of employees in his department only:

```
CREATE VIEW EMP1
AS SELECT LNAME, SALARY, PHONENO
FROM EMP_INFO
WHERE DEPT = 355;
VIEW CREATED
```

Next, the DBA grants *user1* access to the view:

```
GRANT SELECT ON EMP1 TO USER1;
PRIVILEGE(S) GRANTED ON TABLE OR VIEW
```

Here is what the data looks like when *user1* selects from the view:

```
SELECT * FROM DBA.EMP1;

LNAME       SALARY    PHONENO
======      =======   =========
Smith       32654     832
Chen        37666     222

2 ROWS SELECTED
```

The DBA creates the view below so that all employees can look at names and telephone numbers only:

```
CREATE VIEW PHONE
AS SELECT LNAME, PHONENO
FROM EMP_INFO;
VIEW CREATED
```

The DBA then makes the view publicly-accessible:

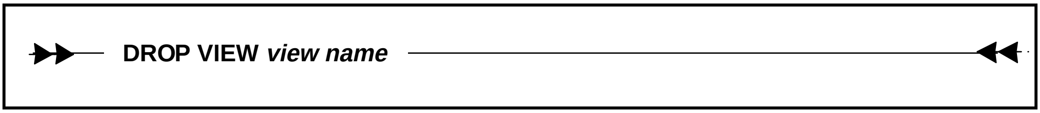
```
GRANT SELECT ON PHONE TO PUBLIC;
PRIVILEGE(S) GRANTED ON TABLE OR VIEW
```

Here is what the data looks like to anyone using the public view:

```
LNAME     PHONENO
======    =======
Smith     832
Jones     285
Brown     721
Joyce     145
Chen      222
Herrlein  185
```

Dropping a view

The DROP VIEW command deletes a specified view.



DROP VIEW *view name*

A view can only be dropped by its creator, SYSADM, or a DBA.

SQLBase automatically drops privileges on the view, and drops any other view that depends partially or wholly on the specified view as well.

Read the *SQL Language Reference* for details about the DROP VIEW command.

SQLTalk examples

The examples below drop the views created previously:

```
DROP VIEW EMP1;
VIEW DROPPED

DROP VIEW PHONE;
VIEW DROPPED
```

Server security

In order to perform administrative operations on a multi-user database server, you must connect to the server and specify its password (if one has been specified in the configuration file (sql.ini)). This prevents unauthorized users from performing destructive operations on the server.

Connecting to a server enables the following operations:

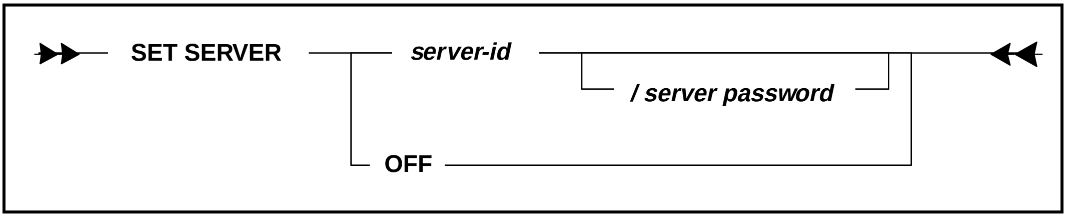
- Backup, recovery, and restoration of a database and its logs
- Creation and deletion of a database and its logs

- Installation and deinstallation of a database

These operations are discussed in *Chapter 4, Databases* and profiled in *Chapter 6, DBA Operations*.

Establishing a server connection

To establish a connection to a server, used SQLTalk's SET SERVER command.



Such strict security is not necessary for a single-user database server. You do not need to use a SET SERVER command before performing administrative operations on a database server.

When you are finished performing administrative operations, use the SET SERVER OFF command to break the server connection.

Read the *SQLTalk Command Reference Manual* for details about the SET SERVER command.

SQLTalk example

The following example invokes SQLTalk in batch mode for the Windows 95 or Windows NT platform with the NOCONNECT and BAT options. This starts SQLTalk and prevents you from being prompted to enter a database name, username, and password. You do not have to be connected to a database in order to establish a connection to the server.

```
SQLTALK BAT NOCONNECT

SET SERVER PROD/SECRET;
SERVER IS SET

... Perform administrative operations

SET SERVER OFF;
SERVER IS OFF
```

Using the SQL/API

In addition to the administrative operations already listed, using the SQL/API enables the following operations as well:

- Aborting a server process
- Terminating the server

The following code fragment shows you how to establish a connection to a database server using the SQL/API:

```
main()
{
   srvname = “SERVER1”;
   password = 0;

/* Connect to the server */

if (rcd = sqlcsv(&handle, srvname, password))
   apierr(“SQLCSV”);
else
   printf(“Connection established to the server\n”);
...
```

The code fragment below shows you how to disconnect from a database server:

```
/* Disconnect from the server */

if (rcd = sqldsv(handle))
   apierr(“SQLDSV”);
else
   printf(“Disconnected from the server\n”);
```

EXECUTE privileges for stored procedures

To grant privileges for other users for stored procedures, use the SQL GRANT EXECUTE ON command. You can grant either your own privileges to other users, or grant them privileges of their own. To revoke privileges, use the REVOKE EXECUTE ON command. Read the *SQL Language Reference* for information on these commands.

External function privileges

When external functions are invoked within stored procedures, follow these rules for granting execute privileges:

- If a user has been granted execute with CREATOR privileges on a stored procedure, then the user does not need EXECUTE privileges on any external function invoked within the procedure. Only the CREATOR of the procedure needs to have EXECUTE privileges on the external functions.

- If a user has been granted EXECUTE with GRANTEE privileges on a stored procedure, the user must also have EXECUTE privileges on an external function invoked within the procedure.

In both cases, you use the GRANT EXECUTE ON command to grant privileges on external functions. This is the same command used for granting privileges on stored procedures.

When granting privileges for external functions with the GRANT EXECUTE ON command, note that the clauses WITH CREATOR PRIVILEGES, and WITH GRANTEE PRIVILEGES do not apply. These clauses only apply to stored procedures.

To revoke external function privileges, use the REVOKE EXECUTE ON command.

Read the *SQL Language Reference* for information on these commands.

System catalog

All authority levels and privileges are recorded these SQLBase system catalog tables.

System Table	Description
SYSCOLAUTH	Contains users' update privileges for individual columns of tables and views.
SYSTABAUTH	Contains users' privileges for tables and views.
SYSUSERAUTH	Contains the database authority level of each user.
SYSEXECUTEAUTH	Contains user's privileges for stored procedures.
SYSOBAUTH	Contains user's privileges for external functions.

Granting access to the system catalog

The system catalog tables are owned by SYSADM.SYSADM. This user can perform the following actions on the system catalog tables:

- Select data
- Create a view, index, or synonym
- Add and drop user-defined columns (with ALTER TABLE) as well as update them

However, no user (including SYSADM) can drop any of the original columns, nor insert or delete rows into or from the system catalog.

By default, SELECT privileges on the system catalog tables are granted to PUBLIC (all users). The exceptions are SYSCOLAUTH, SYSCOMMANDS, SYSEVENTS, SYSTABAUTH, SYSTRGCOLS, SYSTRIGGERS, and SYSUSERAUTH which only SYSADM and a DBA can view. For sensitive data, this may be undesirable. Users may not be able to tell what the actual data is, but they might be able to tell what *type of data* exists.

For this reason, SYSADM or a DBA can create selective views of the system catalog tables and grant users access only to the views. This gives users access to the system catalog data without letting them see everything.

The USER keyword is useful for creating views on system catalog tables because it represents the authorization-id of the current user. When creating a view on a system catalog table, SYSADM can specify USER in the WHERE clause of the SELECT command to restrict the data to only that appropriate for the current user.

For example, the following CREATE VIEW command retrieves data from SYSADM.SYSTABLES only for the tables that the current user owns:

```
CREATE VIEW MYTABLES AS
SELECT * FROM SYSTABLES
WHERE CREATOR = USER;
```

Read *Appendix A, System Catalog Tables* for descriptions of the SQLBase system catalog tables.

Chapter 8

Backing Up and Restoring Databases

This chapter discusses SQLBase's backup and restore options. These topics are covered:

- Backup planning
- Making an online backup of a database
- Making an offline backup of a database
- Restoring an online backup of a database
- Restoring an offline backup of a database
- Backing up and restoring a partitioned database

If you are running a distributed transaction, there are special issues to resolve before restoring a database. Read *Chapter 9, Distributed Transactions* for more information.

Making a backup plan

This section describes how to make a backup plan. You cannot backup and restore snapshots spanning more than one database.

Crash recovery

A database can be damaged in a number of ways such as by a power failure or an operator error in bringing down the server. When an event like this happens, SQLBase tries to restore the database to a consistent state by performing crash recovery automatically when a user connects to a crashed database that has just been brought back online. Crash recovery consists of using the transaction logs to redo any committed transactions which had not yet been written to the database and to undo any uncommitted transactions which were still active when the server crashed.

There are situations where SQLBase will not be able to return a database to a consistent state such as when the transaction logs have been damaged during a media failure.

Use the *fail.sql* file to review information written to this file whether your server crashes or not. This file contains good information and should be reviewed any time any unusual events occur with the database.

Media recovery

Maintenance is a necessary part of a database administrator's job, and involves preparing for events such as a disk head crash, operating system crash, or a user inadvertently dropping a database object. You can recover from media failures and user errors which have damaged a database *if* you make database and log file backups regularly. Making backups of your database and log files from which you can restore the database is the only way you can prevent loss of data.

How often you backup the database and its log files is up to you and depends on how much data you can afford to lose. In general, the following are good guidelines:

- Backup the database once a week.
- Backup the transaction log files once a day.

You can minimize loss of data due to a media failure by backing up transaction logs frequently. You should backup all logs since the last database backup so that in the case of a media failure they can be used to recover the database up to the point of that last log backup.

In addition, you should save the database and log files from the last several sets of backups taken. For example, if you make a BACKUP SNAPSHOT every Sunday, and make log backups every night, a backup set would consist of the snapshot, and Monday through Saturday's log file backups. Never rely on just one backup!

Note: Never delete transaction log files. SQLBase automatically deletes log files either when they are backed up or when they are no longer needed for transaction rollback or crash recovery, depending on whether LOGBACKUP is on or off. A database file may be useless without its associated log files.

To ensure a relatively fail-safe strategy for backing up and restoring the database to log files, retain two cycles of backups as follows:

Cycle 1Day 1 backup DBS and log files

Day 2 backup log files

...

Day n backup log files

Cycle 2Day 1 backup DBS and log files

Day 2 backup log files

Cycle 3:... (Repeat process)

Only when Cycle 3 is successful, should cycle 1 be eliminated.

Also, note that Day n for the previous cycle should coincide with Day 1 of the next cycle to minimize loss of data.

Recovery commands

SQLTalk commands for recovery are:

- SET RECOVERY ON/OFF

  Recovery is on by default. This means that SQLBase logs all before- and after-images of changes to the database and inserts log records for transaction control (checkpoints, for example). As part of transaction control, some log records document when a transaction started and how it ended (COMMIT and ROLLBACK, for example). Transaction logs enable transaction rollback, crash recovery, and media recovery.

  Recovery should only be turned off if you can afford to lose your data in the event of user error, a system crash, or a media failure. Once you set recovery off, some options are reset to their defaults including: autocommit, bulk execute, isolation level, load version, cursor-context preservation, restriction, rollback, scroll, and timeout.

  You should back up your database before turning recovery off.

- SET LOGBACKUP ON/OFF (*sqlset* with the SQLPLBM parameter)

  If media recovery is important to your site, the LOGBACKUP parameter should be set on, specifying that logs are to be backed up by the DBA before SQLBase deletes them.

  By default, LOGBACKUP is not enabled and SQLBase deletes log files as soon as they are not needed to perform transaction rollback or crash recovery. This is done so that log files do not accumulate and fill up the disk. If LOGBACKUP is off, you will not be able to recover the database if it is damaged by a user error or a media failure.

  This option is database-specific and needs to be set on only *once*. The setting will stay active until changed. You do not need to set the option each time a database is brought back online. Resetting this option affects whether log files are deleted or saved for archiving. To avoid gaps in your log files, set this option once to on.

  LOGBACKUP must be on to do a BACKUP DATABASE or BACKUP LOGS, but does not need to be on to do a BACKUP SNAPSHOT.

  The default is off.

- SET CHECKPOINT (*sqlset* with the SQLPCTI parameter)

  The checkpoint time interval parameter controls how often a recovery checkpoint operation is done. The default is every minute. Depending on the applications running against the server, a checkpoint operation can affect performance. Increasing the checkpoint interval may provide better performance. Longer checkpoint intervals reduce the impact on performance, but increase the time taken to perform crash recovery.

Backing up a database

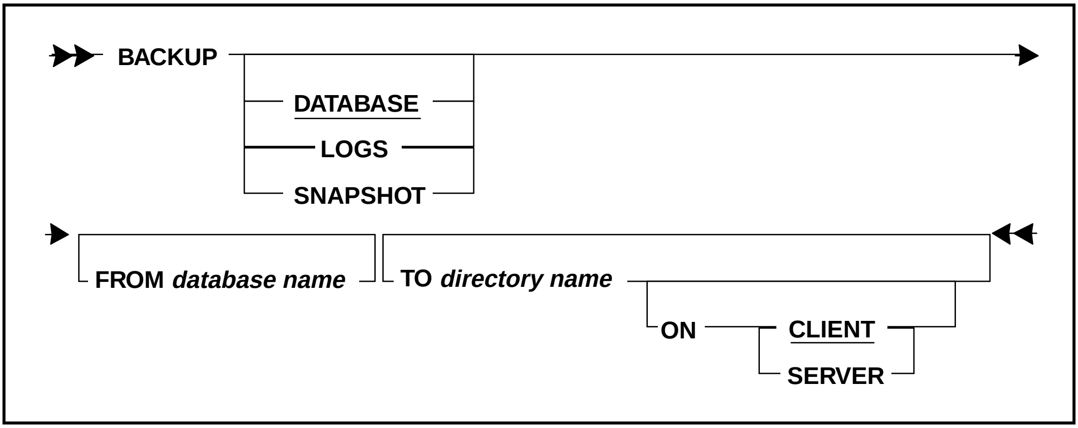
There are two ways of backing up a database: online and offline.

- An online backup is a copy of a database (*.dbs*) file and its log (*.log*) files that you make with a SQLTalk BACKUP command or a SQL/API function while the server program is running (users are connected to the database and transactions are in progress).
- An offline backup is a copy of the database file and log files that you make with an operating system utility or command (such as *copy*) after successfully bringing the server down.

Online backups

The advantage of an online backup is that users can access the database while the backup is being done. This is important to sites which require the database to be up 24 hours a day.

The BACKUP command backs up a database, its transaction log files, or both.



Syntax diagram of the BACKUP command

Note: When using the BACKUP command, both partitioned and non-partitioned databases cannot exceed 2 gigabytes. If you do have a database greater than 2 gigabytes, you must perform the BACKUP command to multiple segments. This ability is provided in the BACKUP DATABASE option using the FROM and TO clauses. See “Creating a segmented backup” in the following section.

The online backup options include:

- BACKUP SNAPSHOT (*sqlbss*)

  Backs up only the database file and those log files needed to restore the database to a consistent state. This includes the current active log file since BACKUP SNAPSHOT forces a log rollover.

  This command is the only BACKUP command which does not require LOGBACKUP to be on. If LOGBACKUP *is* on, the log files left in the database directory should be backed up with a BACKUP LOGS command. SQLBase will then delete them automatically.

- BACKUP DATABASE (*sqlbdb*)

  Backs up the database file. You should *never* back up a database without also backing up the log files with it.

With this option, you can perform a segmented backup of your database. To do this, you create a control file that describes the location and size of the segments to which you want to backup your database, and specify the control file in the FROM and TO clauses. Each segment must be 2 gigabytes or less in size. For details, read the following section Creating a segmented backup.

- BACKUP LOGS (*sqlblf*)

  Backs up the log files and then deletes them.

BACKUP SNAPSHOT is the recommended way to backup your database and log files because it is easy and provides you with a backup from which you can recover the database in one step.

BACKUP DATABASE and BACKUP LOGS are provided for sites with large databases that wish to do incremental backups. Between database backups (both BACKUP SNAPSHOT and BACKUP DATABASE), you should back up log files using the BACKUP LOGS command or the *sqlblf* API function. For example, you could back up the database and logs every Sunday, while on Monday through Saturday you could back up the logs.

A backup directory can be on a client or server computer. Once you have backed up a database and its log files to a directory, you can copy the backup files to archival media and delete the backup files from the client or server disk.

Online backups briefly lock the database during two periods: when the backup begins and the main control page of the database is updated, and when the backup is almost completed and another update takes place.

The examples later in this section show how to make and restore online backups.

Creating a segmented backup

To perform segmented backups, you must create a control file (*databasename*.BCF) that describes the location and size of the segments to which you want to backup your database. To create a control file, use any editor on your system. The file must be in ASCII format.

The BACKUP command requires the directory where the control file resides in the TO and FROM clause. If SQLBase detects a control file is present, a segmented backup operation is automatically performed. If the control file is not present, SQLBase backs up the database to a single *databasename*.BKP file (if the file is less than 2 gigabytes).

The backup control file follows this syntax:

```
FILEPREFIX <filename prefix>
DIR <destination dir> SIZE <file segment size>
DIR <destination dir> SIZE <file segment size>
```

This file provides the following information:

Parameter	***Description***
FILEPREFIX	The prefix of the file segment names used for the load.
DIR	The destination directory where the load file segments reside.
SIZE	The maximum size of the file segment in megabytes. You can specify a null or integer value of 1 through 2048 megabytes. The control file must indicate a minimum aggregate size to account for all the backup data. Use the following to calculate the maximum number of bytes you can allocate: bytes = (size * 1048576) which is the maximum file size common across most systems. The last unload file segments may not use the entire size that you allocated.

The following example shows a load control file located on a Windows NT server:

Example:

```
FILEPREFIX  dbs
DIR  c:\unldir\ SIZE 5
DIR  d:\unldir\ SIZE 4
DIR  e:\unldir\ SIZE 10
```

In this example, there are three file segments with the following characteristics:

- a segment called **c:\unldir\dbs.1** with 5 megabytes
- a second segment called **d:\unldir\dbs.2 with 4 megabytes**
- a third segment called **e:\unldir\dbs.3** with 10 megabytes

The name of this control file itself is *dbs.bcf*. SQLBase backs up the information from these three segments according to their listed order.

Note: If you are using a NetWare Server, be sure to specify both the volume mapping letter and the volume name for the file segments.

Offline backups

The advantage of an offline backup is that you can back up files directly to archival media. A SQLBase BACKUP command will not back up files to a tape drive, for example.

Before you can make an offline backup, you must shut down the server gracefully. To shut down a server when using a multi-user server program like *dbntsrv.exe*, you must press **Esc** or deinstall the database with the SQLTalk DEINSTALL command or the *sqlded* API function.

You make an offline backup using an operating system command or utility. Below is an example of an offline backup done using the COPY command:

```
C> COPY C:\CENTURA\MYDB.DBS C:\BACKUPS\MYDB.BAK
C> COPY C:\CENTURA\1.LOG C:\BACKUPS
C> COPY C:\CENTURA\2.LOG C:\BACKUPS
```

Follow an offline backup with SQLTalk’s SET NEXTLOG command to tell SQLBase that an offline backup of one or more log files has occurred. SQLBase now knows that these backed up log files are candidates for deletion. If you had backed up the log files with SQLBase's online BACKUP command, the files would have been automatically deleted. In the above case, the NEXTLOG command would be:

```
SET NEXTLOG 3;
```

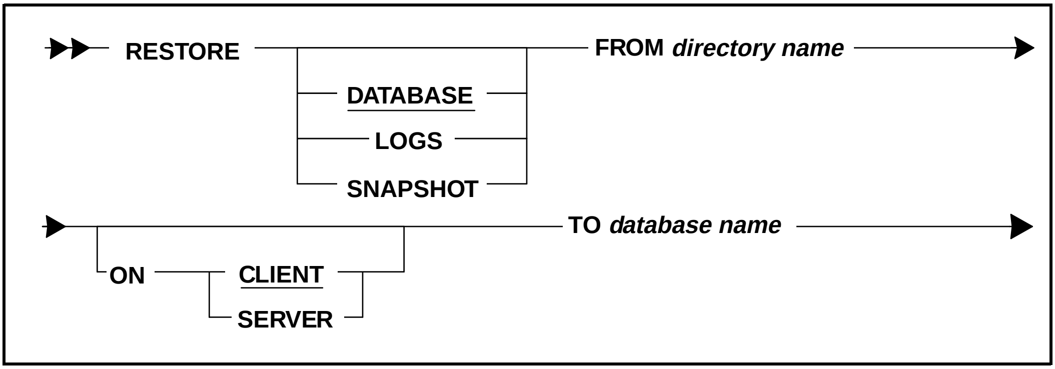
Restoring a database

Data restoration is the process of recovering data which has been lost. Recovery of data is dependent on the DBA's backup planning and the frequency of backups made.

Restoring an online backup

Users cannot be connected to a database during a restore and recovery process. You should deinstall a multi-user database with SQLTalk’s DEINSTALL DATABASE command or the SQL/API’s *sqlded* function, perform the restore and rollforward, and then re- install the database with SQLTalk’s INSTALL DATABASE command or the SQL/API *sqlind* function.

If a database becomes damaged, you can restore a database backup with SQLTalk’s RESTORE command or the SQL/API’s *sqlrss* function.



Syntax diagram of the RESTORE command

Note: When using the RESTORE command, non-partitioned databases cannot exceed 2 gigabytes. If you do have a database greater than 2 gigabytes, you must perform the RESTORE command from multiple segments. This ability is provided in the RESTORE DATABASE option using the FROM clause. Read the following section Restoring from a segmented backup.

The restore options include:

- RESTORE SNAPSHOT (sqlrss)

  After executing a RESTORE SNAPSHOT command or calling the *sqlrss* function, no further action is necessary because the command copies not only the backup database file but also the backup log files to the database subdirectory.

- RESTORE DATABASE (*sqlrdb*)

  If you did not make the backup with BACKUP SNAPSHOT or you did a BACKUP SNAPSHOT and want to roll forward from that point to recover as much work as possible, you can give a RESTORE DATABASE command or call the SQL/API *sqlrdb* function in a program.

  You must use this option if you are restoring a database from backup segments. This is required if you have a non-partitioned or partitioned database greater than 2 gigabytes. Each segment must be 2 gigabytes or less in size.

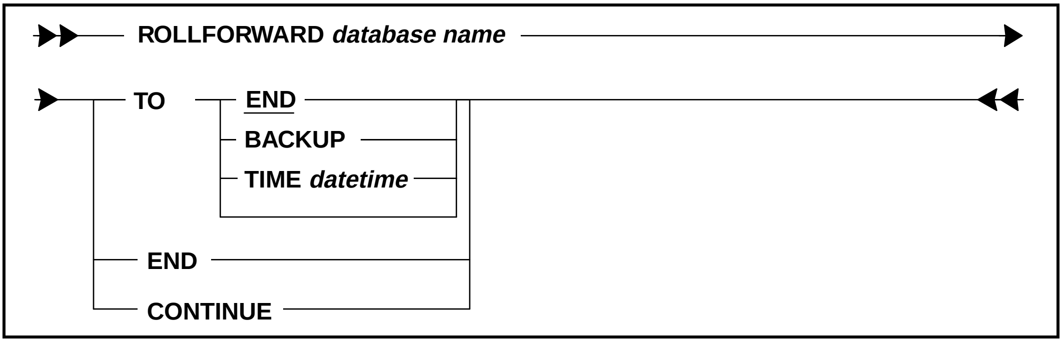
Restoring from a segmented backup

To perform a restore from a segmented backup, you must specify a control file (*databasename*.BCF) that was used to create a successful segmented database backup. This control file describes the location and size of the segments from which you want to restore your database. For details on the backup control file, read *Creating a segmented backup* on page *8-6*.

If SQLBase detects a control file is present, a segmented restore operation is automatically performed from the backup segments specified in the control file. If the control file is not present, SQLBase expects to restore the database from a single backup file that you specify named *databasename*.BKP.

Using ROLLFORWARD

To rollforward changes made after the database backup and bring the database up-to-date, use SQLTalk's ROLLFORWARD command or the SQL/API's *sqlrof* function.



Syntax diagram of the ROLLFORWARD command

The rollforward options are:

- ROLLFORWARD TO END - Roll forward through all log files available (the default). This recovers as much of the user's work as possible.
- ROLLFORWARD TO BACKUP - Roll forward to the end of the backup restored. This recovers all committed work up to the point when the database backup was completed. This is equivalent to a RESTORE SNAPSHOT.
- ROLLFORWARD TO TIME - Roll forward to a specified date and time. This allows you to recover a database up to a specific point in time, and in effect rolls back large "chunks" of committed and logged work that you no longer want applied to the database. For example, if data is erroneously entered into the database, you would want to restore the database to the state it was in before the bad data was entered.

You must have backed up *all* the database's log files and must apply them in order or the rollforward will fail. If you are missing any of the log files, you will not be able to continue rolling forward from the point of the last consecutive log. For example, if you have *1.log*, *2.log*, *4. log*, and *5.log*, but *3.log* is missing, you will only be able to recover the work logged up to *2.log*. *4.log* and *5.log* cannot be applied to the database. An unbroken sequence of log files is required to recover a database backup to its most recent state.

The rollforward operation stops if SQLBase cannot find a log file that it needs. In this situation, a RESTORE LOGS command will copy the log files needed from the backup directory to the database directory.

If there are more logs to be processed than can fit on disk at one time, you can give the RESTORE LOGS command repeatedly to process all the necessary logs.

If a log file requested is not available, you can enter a ROLLFORWARD *database* END command to end recovery using the data restored up to that point. The *sqlenr* API function does the same thing.

Restoring an offline backup

Recovering an offline backup is done in one of two ways:

- If the backup consists of only a database file, restore it by copying it over the existing damaged database file, making sure the extension is *.dbs* (you may have changed it, for example, to *.bkp* when you backed it up), and then connecting to the database. All changes made since the offline backup was done will be lost.
- If the backup consists of a database file and one or more log files, use the SQLTalk RESTORE DATABASE command or *sqlrdb* API function to restore the database and then give a ROLLFORWARD command to apply the logs to bring it up-to-date. The RESTORE copies the backup to the database subdirectory, and the ROLLFORWARD applies the committed and logged changes made to the database since the offline backup of the database was taken.

  If SQLBase cannot find the log files to rollforward, you can restore them by either a RESTORE LOGS command (which automatically does a ROLLFORWARD CONTINUE) or with a copy command or utility, and then give a ROLLFORWARD CONTINUE command explicitly or call the *sqlcrf* API function to apply the log files.

In order for the RESTORE command to work, the name of the database backup file must be *database*.BKP.

Examples

This section provides backup and restore examples.

Example 1 - Snapshot

The following example shows you the recommended way to back up (BACKUP SNAPSHOT) and restore (RESTORE SNAPSHOT) a database and its logs. Snapshots are easy because you can backup a database or restore it with one command.

Recall that BACKUP SNAPSHOT is the only backup command which does not require LOGBACKUP to be on.

The first step is to establish a connection to the server:

```
SET SERVER SERVERP/PPPPP;
SERVER IS SET
```

Next, create a subdirectory on the client where you will store the backup:

```
$'MKDIR \BACKUP';
$'MKDIR \BACKUP\DEMOX';
```

Then back up the *.dbs* and *.log* files to the client computer:

```
BACKUP SNAPSHOT TO \BACKUP\DEMOX ON CLIENT;
SNAPSHOT BACKED UP
```

Now you are ready to restore the database and its log files. Remember that you cannot perform a *restore* operation while any user (including yourself) is connected to the database:

```
SHOW CONNECT;
*****
Username: SYSADM
Database: DEMOX
Cursor number: 1
*****
```

Disconnect from the database you are restoring and connect to another database temporarily:

```
CONNECT EMP 2;
CURSOR 2 CONNECTED TO EMP
USE 2;
DISCONNECT 1;
CURSOR 1 DISCONNECTED
```

Now restore the database and its log files from the client computer:

```
RESTORE SNAPSHOT FROM \BACKUP\DEMOX ON CLIENT TO DEMOX;
SNAPSHOT RESTORED
```

Example 2 - Backing up and restoring database and log files separately

This example shows how to backup and restore a database and its log files separately. You should *always* backup the transaction log files as well as the database. Remember that LOGBACKUP must be on.

The first step is to establish a connection to the server:

```
SET SERVER SERVERP/PPPPP;
SERVER IS SET
```

Next, create a subdirectory on the client where you will store the backup:

```
$'MKDIR \BACKUP';
$'MKDIR \BACKUP\DEMOX';
```

Now back up the database and log files to the client computer:

```
BACKUP DATABASE TO \BACKUP\DEMOX ON CLIENT;
DATABASE BACKED UP
```

Force a log rollover:

```
RELEASE LOG;
RELEASE LOG COMPLETED
```

```
BACKUP LOGS TO \BACKUP\DEMOX ON CLIENT;
3 LOGS BACKED UP
```

Now you are ready to restore the database and its log files. Remember that you cannot perform a *restore* operation while any user (including yourself) is connected to the database:

```
SHOW CONNECT;
*****
Username: SYSADM
Database: DEMOX
Cursor number: 1
*****
```

Disconnect from the database you are restoring and connect to another database temporarily:

```
CONNECT EMP 2;
CURSOR 2 CONNECTED TO EMP
USE 2;
DISCONNECT 1;
CURSOR 1 DISCONNECTED
```

Now restore the database from the client computer and apply the transactions in the log files to the database using the ROLLFORWARD command:

```
RESTORE DATABASE FROM \BACKUP\DEMOX TO DEMOX;
DATABASE RESTORED
```

Use the ROLLFORWARD command to initiate recovery for this restored database.

```
ROLLFORWARD DEMOX;
ROLLFORWARD STARTED

   The log file 1.LOG could not be found. Use the RESTORE
LOGS command to restore this log and continue the
rollforward process. If this log is not available, use the
ROLLFORWARD <database> END command to complete the recovery
process.

RESTORE LOGS FROM \BACKUP\DEMOX TO DEMOX;
3 LOGS RESTORED

   The log file 4.LOG could not be found. Use the RESTORE
LOGS command to restore this log and continue the
rollforward process. If this log is not available, use the
ROLLFORWARD <database> END command to complete the recovery
process.

ROLLFORWARD DEMOX END;
ROLLFORWARD COMPLETED
```

Backing up and restoring partitioned databases

You can back up large partitioned databases using SQLTalk's BACKUP, RESTORE, and ROLLFORWARD commands. If your database is greater than 2 gigabytes, you need to create a segmented backup database. Refer to the sections *Creating a segmented backup* and *Restoring to a segmented database* in this chapter for details.

Online backup

Following is an example for converting a non-partitioned database to a partitioned database using the online BACKUP and RESTORE commands.

1. Back up the current DEMOX database. This step assumes you have already entered a SET SERVER command.

   ```
   BACKUP DATABASE TO \BACKUP\DEMOX;
   DATABASE BACKED UP
   ```

2. Drop the old DEMOX database.

   ```
   DROP DEMOX;
   ```

3. Create the new backup database.

   ```
   CREATE DATABASE DEMOX;
   ```

   Note that in this step, you may need to create a partitioned database. For details, read *Partitioning a database* on page *6-1*

4. Restore the DEMOX database.

   ```
   RESTORE DATABASE FROM \BACKUP\DEMOX TO DEMOX;
   DATABASE RESTORED
   ```

Note: When backing up and restoring a partitioned database, you also need to backup and restore the MAIN database. For details, see the *The MAIN database* on page *8-15*.

Offline backup

You can also create an offline backup of a partitioned database. To do this, first be sure to shut down the server gracefully. Then use your operating system command to back up the dbareas that make up your partitioned database. Also be sure to make backups of the MAIN database and its logs. The MAIN database is described in the next section.

If you want to restore an offline backup, you can restore all relevant dbareas and the MAIN database, including its files, in the appropriate locations.

The MAIN database

The MAIN database contains information about the partitioned databases. Because it is small, it is easiest to back it up online.

Note: *Always* back up the MAIN database when you back up a partitioned database.

To restore the MAIN database, first disable partitioned databases with the following command:

```
SET PARTITIONS OFF;
```

Once recovery is complete, re-enable partitioned databases with the following command:

```
SET PARTITIONS ON;
```

You must restore the MAIN database by rolling forward through all the log files that were backed up when the MAIN database itself was backed up. You cannot do a partial recovery of the MAIN database.

Do not delete the MAIN database.

Chapter 9

Distributed Transactions

This chapter describes how to create distributed transactions. Distributed transactions in SQLBase use the standard two-phase commit protocol. This protocol maintains transaction integrity and communication among the individual databases involved in a distributed transaction.

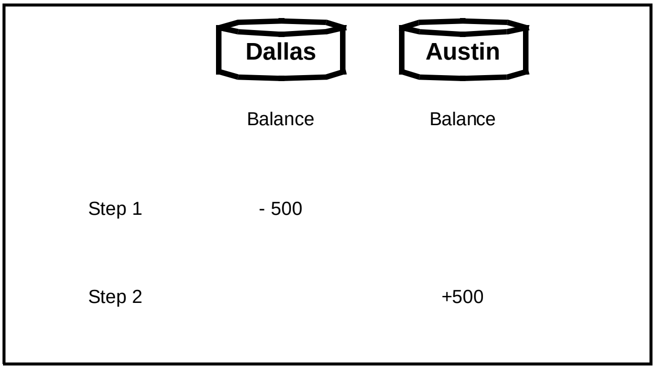
What is a distributed transaction?

A distributed transaction coordinates SQL statements among multiple databases that are connected by a network. The databases that participate in a distributed transaction can reside anywhere on the network.

In a distributed transaction, the coordinating application communicates among the participant databases and verifies data integrity. It maintains this integrity even when a crash occurs.

A distributed transaction conforms to the same data consistency rules as a single database transaction — either all of the transaction's statements commit, or none at all.

A distributed transaction can be necessary for efficient transaction management. For example, assume a bank in Dallas needs to transfer $500 to its Austin branch.



The application uses the following steps:

1. Debit $500 from the Dallas branch.
2. Credit the $500 to the Austin branch.

It is possible to use regular transactions here by just sending a COMMIT message to both branches after step 2 and waiting for their replies. However, severe data integrity problems could result if a system/network error prevents either site from completing their part of the transaction:

- If a network or system failure occurs at the Dallas site after step 1 but before step 2, the $500 would not be debited. Unaware of this problem, the Austin branch would commit its $500 deposit. The customer would end up with two deposits.
- If a network or system failure occurs at the Austin site before step 2, the deposit would not be credited to the Austin branch. Unaware of the problem,

Dallas would commit the $500 debit. The bank would have no record of the customer's deposit at all.

A distributed transaction using two-phase commit eliminates these data integrity problems by coordinating communication between sites throughout the various transaction stages.

Setting up a distributed transaction

You can set up a distributed transaction either in SQLTalk or with a SQL/API function. You can also create a distributed transaction through Centura's Team Developer product, although that is not documented in this chapter.

Server connects *(sqlcsv)* and connects with recovery turned off cannot participate in a distributed transaction. In addition, an application cannot connect to a database in both distributed and non-distributed transaction mode.

In a distributed transaction, one of the participating database servers must also be the *commit server*. The commit server logs information about the distributed transaction and assists in recovery after a network failure. To enable commit server capability for a server, set the *commitserver* keyword to 1 (on) in sql.ini. Read *Components* on page *9-7* for more information on the commit server.

Databases participating in a distributed transaction must conform to the following communication requirements:

- They must reside on the same network. They can reside on the same server, but this is not required.
- Each participating database server that has commit service enabled must be able to connect to all other servers involved in the distributed transaction. If all the servers have commit service capability, they all must be able to connect with each other.
- If you are using Novell's NetWare, specify the [nwclient] section for each server that is participating in a distributed transaction. This allows servers to communicate mutually. Communication between servers only occurs when a commit server:
  - Verifies it can talk to all other participating servers at the time of a distributed commit. (This is performed at most once per participant.)
  - Attempts to contact other participating servers under a failure condition.

Using SQLTalk

In SQLTalk, set the DISTRANS option on to start a distributed transaction. Any new database you connect to is then part of a distributed transaction.

The following example sets up a distributed transaction between the Austin and Dallas sites in a sample SQLTalk session:

Notice that you need to disconnect the first cursor in order to use it in the distributed transaction, since it was initially connected in a regular transaction.

```
set distrans on;
DISTRIBUTED TRANSACTION MODE IS NOW ON
connect austin 2;
CURSOR 2 CONNECTED TO AUSTIN
disconnect 1;
CURSOR 1 DISCONNECTED
connect dallas 1;
CURSOR 1 CONNECTED TO DALLAS
select * from account where account_num=14560;

NAME                        ACCOUNT_NUM   BALANCE
=========================== ===========   ==========
Nanda Smith                14560        1000

1 ROW SELECTED

update account set balance=balance-500 where account_num=14560;
1 ROW UPDATED
use 2;
select * from account where account_num=14560;

NAME                        ACCOUNT_NUM   BALANCE
=========================== ===========   ==========
Nanda Smith                14560        1000

1 ROW SELECTED

update account set balance=balance+500 where account_num=14560;
1 ROW UPDATED
commit;
TRANSACTION COMMITTED
use 1;
select * from account where account_num=14560;


NAME                       ACCOUNT_NUM      BA
=========================== =============     =========
Nanda Smith                        14560         500

1 ROW SELECTED
```

Note here that only one COMMIT statement is issued. In a regular transaction, you would need to issue two COMMIT statements, one on each database.

```
use 2;
select * from account where account_num=14560;
NAME                          ACCOUNT_NUM      BALANCE
=========================== =============     =========
Nanda Smith                           14560         600

1 ROW SELECTED
exit;
```

Using the SQL/API

In the SQL/API, use the *sqlset* function in conjunction with the SQLPDTR parameter to set distributed transaction mode on. Once you set this parameter on, all subsequent commands automatically become part of a distributed transaction. Read the *Application Programming Interface* manual for an example.

Other issues

This section describes how creating distributed transactions affects SQLBase in general.

Backing up and restoring

You cannot back up and restore database snapshots spanning more than one database.

Isolation

Changing the isolation level of a distributed transaction changes it for all participating databases. Also, since changing isolation levels results in an implicit commit, SQLBase issues a coordinated commit when you are using a distributed transaction.

Lock wait timeout

To set a specified lock wait timeout for a distributed transaction, use the *sqlset* function with the SQLPWTO parameter. Setting the timeout for a distributed transaction sets it for all participating cursors. This is the mechanism to get out of a global lock in a distributed transaction.

Savepoints

When you set a savepoint in a distributed transaction, it applies to all the databases which participate in that transaction. For example, assume that the Austin database contains a savepoint. A rollback to the savepoint rolls back actions on both the Austin and Dallas databases.

```
set distrans on;
DISTRIBUTED TRANSACTION MODE IS NOW ON
connect austin 2;
CURSOR 2 CONNECTED TO AUSTIN
```

```
              disconnect 1;
              CURSOR 1 DISCONNECTED
              connect dallas 1;
              CURSOR 1 CONNECTED TO DALLAS

(1).    →     insert into account values.....
              use 2;+
              set savepoint savpoint1;
              use 1;
              insert into account values ....
(2)     →     select * from account where account_num=14560;
              1 ROW SELECTED
              use 2;
              insert into account values ...
(3)     →     rollback savpoint1;
              commit;
```

In this scenario, statements 2 and 3 are rolled back, but statement 1 is committed.

AUTOCOMMIT

Turning on AUTOCOMMIT for a cursor in a regular transaction causes an implicit commit after each command. For a cursor in a distributed transaction, AUTOCOMMIT causes an implicit coordinated commit every time that cursor gets executed.

Last disconnect

For regular transactions, the last disconnect from a database results in an implicit commit. For distributed transactions, however, the last disconnect from any participating database returns an error if there is any uncommitted work, such as modifications that have been neither committed nor rolled back.

Recovery

You cannot set recovery off in a distributed transaction.

Two-phase commit

The two-phase commit protocol coordinates a transaction commit process on all participating databases. When a network or system failure occurs, the transaction is resolved rather than left in a partially committed state. Without two-phase commit, you could end up with a partially committed distributed transaction, which is committed on one system, and rolled back on another.

Understanding the basic mechanics of a two-phase commit can help you plan efficient database access, especially if you need to manually resolve a transaction if a server crashes.

Components

Two-phase commit uses the following components:

- coordinator
- participant
- commit server

Coordinator

The application that initiates a distributed transaction is called the *coordinator*. The coordinator is responsible for starting the two-phase commit, communicating among the participants, and logging transaction information with the commit server.

The actual code for the coordinator resides in Centura's SQL/API, not in the coordinating application. You do not need to write any code yourself to enable coordinator responsibilities.

In the bank example, the transfer program is the coordinator.

Participant

The databases involved in the transaction are called *participants*, and can exist at different sites. In the bank example, the participants are the Dallas and Austin branches.

The participants should either reside on the same network, or be able to communicate across the network. This is important in a failure recovery situation, which is discussed later in the chapter.

Each participant has its own SYSADM.SYSPARTTRANS table, where it records information about the status of the transaction. This table contains information about transactions which are *in-doubt* (in an undecided state) after crash recovery or data restoration. There is one row for each in-doubt transaction. Once a transaction is resolved, the information is removed from the table. See the following section on *Failure recovery* for more information on in-doubt transactions.

Commit server

Two-phase commit uses a *commit server* that logs information about each distributed transaction. It performs the following functions:

- Assigns a global ID to the transaction. This global ID distinguishes a distributed transaction from a regular transaction.
- Starts and ends a distributed transaction.
- Commits or rolls back a distributed transaction.
- Logs the status of a distributed transaction throughout its various stages.

- Assists in recovery in case of a crash.

The commit server logs information about the distributed transaction in two MAIN database tables:

- SYSADM.CSVTRANS

  This table records general information about a transaction such as the global ID, number of participants, and current state of the transaction.

- SYSADM.CSVPARTS

  This table contains information about all the participants in a transaction, such as database name and user name.

The information in these two tables help in the recovery process if your system crashes. This is described in the *Failure recovery* section later in this chapter.

Note: If a database server is designated to be the commit server for a distributed transaction, a MAIN database is created for that server even if it is non-partitioned.

The commit server only retains information about a transaction while it is being processed. After a transaction is resolved (COMMIT or ROLLBACK), the commit server deletes all information about the transaction from the two systems tables.

You cannot create a distributed transaction unless one of the participating database servers has commit service enabled. To enable commit server capability for a server, set the *commitserver* keyword to 1 (on) in the database server's sql.ini file.

Generally, there is only one participant database server with commit service enabled. However, if the distributed transaction connects to multiple database servers that have commit service capability, the coordinator randomly chooses one to be the commit server for the transaction.

Process

There are two stages in the two-phase commit process: prepare and commit. In the first stage, the coordinator sends a PREPARE message to all the participants. Each participants decides individually whether it can commit the transaction, or needs to rollback.

If the participant cannot commit the transaction, it rolls back its part of the transaction, and sends a ROLLBACK vote to the coordinator. If the coordinator receives even one ROLLBACK vote, it instructs the other participants to rollback the transaction.

If the participant can commit the transaction, it sends a COMMIT vote to the coordinator. Once a participant votes to commit, it cannot later decide to rollback the transaction.

If the coordinator receives COMMIT votes from all the participants, it sends them all an instruction to go ahead and commit the transaction. The participants commit the transaction accordingly.

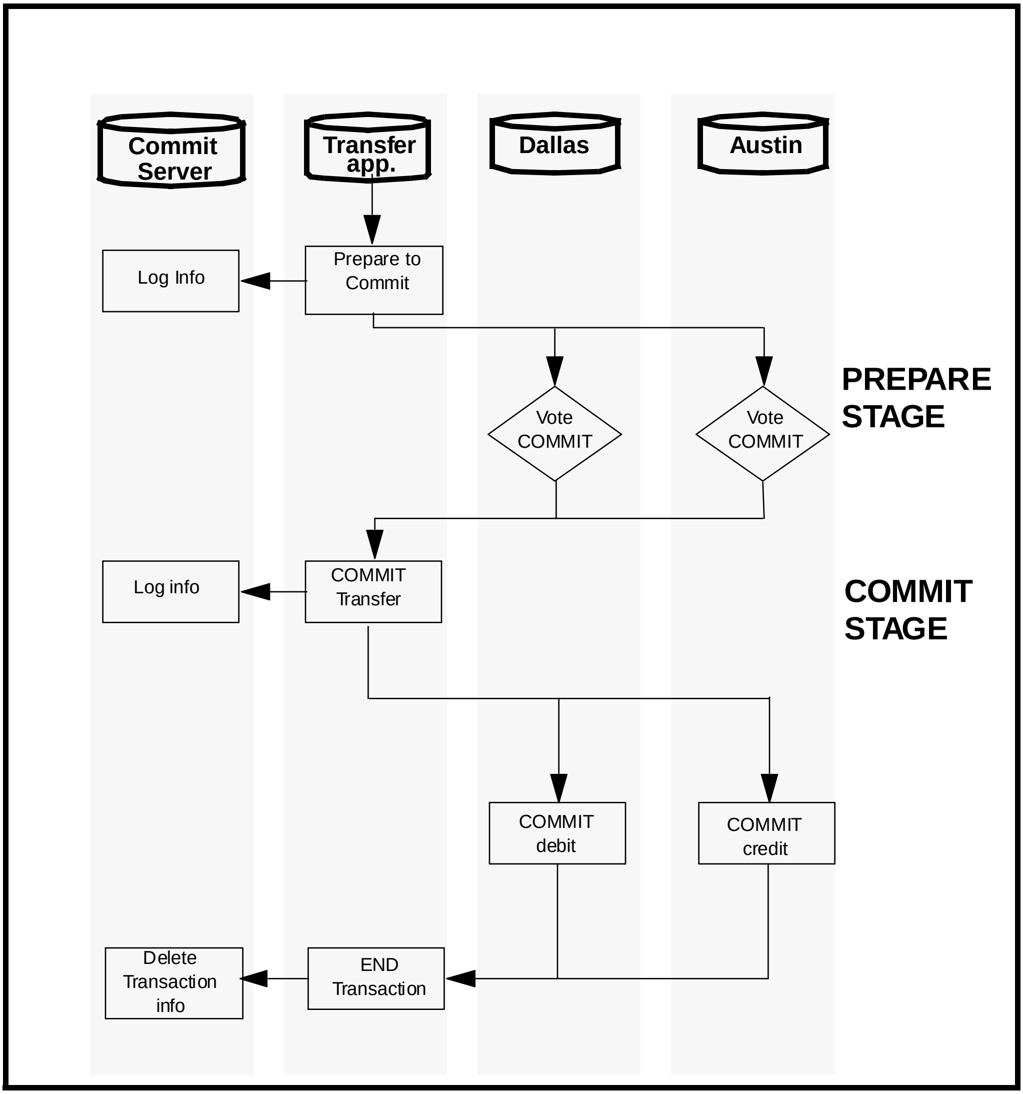
The Commit Server logs information about the transaction during the two-phase commit process.

After a commit or rollback, the system writes a commit log record and releases all the transaction's locks. The transaction entry is also removed from SYSADM.CSVTRANS, SYSADM.CSVPARTS, and the SYSADM.SYSPARTTRANS tables in the participating databases.

The following figures use the bank example to demonstrate the two-phase commit process. The first figure shows a successfully committed transaction; the second demonstrates what happens when one of the participants aborts.

COMMIT Example

In this figure, the participants (Dallas and Austin branches) successfully commit the transfer process. The transfer application is the coordinator.



1. In the PREPARE stage, the coordinator (transfer program) sends a message to the Dallas and Austin databases to prepare a commit.

   Both Austin and Dallas vote to commit the transaction. They are now in an in-doubt state until they actually receive the COMMIT message from the coordinator.

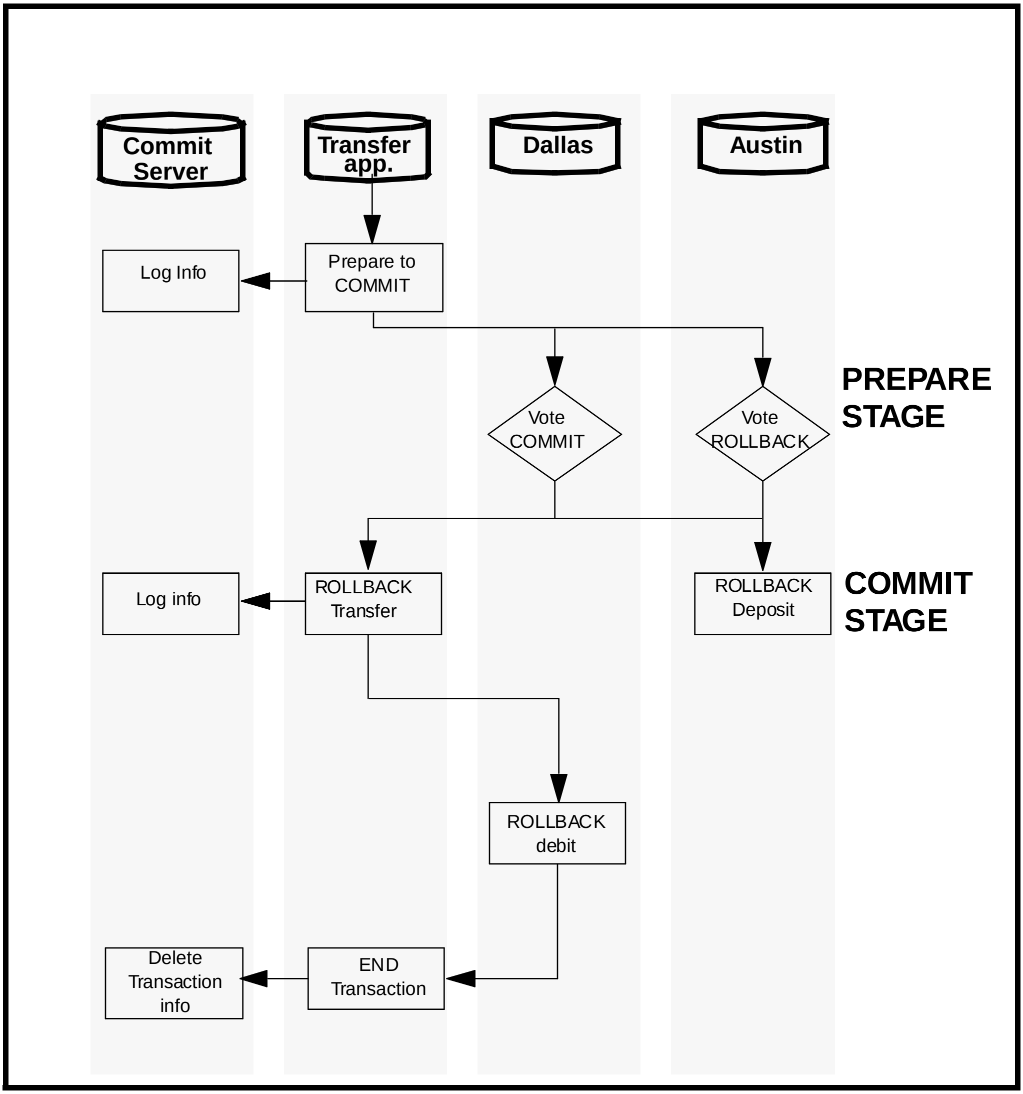
2. Since it received commit votes from all the participants, the coordinator begins the commit process. It sends a COMMIT message to both Austin and Dallas. Austin and Dallas both commit their part of the transaction.

   When they complete the COMMIT, both branches inform the transfer program, and delete information about the transaction in their local SYSADM.SYSPARTTRANS table. The program logs this information with the commit server. The commit server then deletes the information about the transaction in both SYSADM.CSVTRANS and SYSADM.CSVPARTS in the MAIN database.

For a distributed transaction, a commit operation results in a coordinated commit across all the databases. This is also true for an implicit commit.

ROLLBACK example

In this example, the Austin branch is unable to commit the transfer. The following figure demonstrates this process.



In this example, Austin cannot commit the transaction, and votes to rollback. The coordinator accordingly sends a ROLLBACK message to Dallas.

A rollback operation rolls back the changes to all the databases, including implicit rollbacks. For example, if a transaction needs to be rolled back due to an error, the rollback is performed on all the databases to which that transaction is connected.

Failure recovery

A network or system failure can cause special problems for databases using distributed transactions, such as partially committed transactions, locking of needed data, and data inconsistency.

In-doubt transactions

If the database or server crashes, the database is recovered when you reconnect. At this point, there may be in-doubt transactions that need to be resolved. A transaction is in-doubt when it is ready to commit a transaction, but has not yet received a COMMIT or ROLLBACK message from the coordinator; it is unknown which of the transactions committed and which did not. To resolve these transactions automatically when you reboot, SQLBase uses the *commit server daemon*.

For example, assume that the bank transfer program crashes after Dallas commits its delete command, but before Austin records the transfer. The Dallas transaction is resolved, but the Austin transaction is in-doubt. It remains in-doubt until the commit server daemon can resolve it after you reboot the system.

Manually resolving transactions

You should always allow SQLBase to automatically recover in-doubt transactions. However, if the network connections are damaged, or the server will take too long to recover, occasionally you may need to manually commit or rollback the transaction if it is locking data required by other users, or pinning important log files.

Note: If you do decide to manually override a transaction, first consult with the administrators at the other locations. Be aware that if you make a wrong decision, you may create data inconsistencies that must be manually corrected, and may be difficult to trace.

Use the following guidelines for resolving in-doubt transactions yourself:

1. Find out which transactions are in-doubt.
2. Find the status of these in-doubt transactions if possible.
3. Force the resolution.

Finding in-doubt transactions

To locate in-doubt transactions query the local SYSADM.SYSPARTTRANS table. Note the global ID numbers of the transactions listed, and the name of the commit server.

It is important to remember that this table does not contain information about all current in-doubt transactions while the database server is active. You only find out about all the transactions when you reconnect after the crash.

Finding transaction status

If only the network connection crashed and you can still access the commit server locally, locate the commit server and try to access it. Then, use the global ID from SYSADM.SYSPARTTRANS to find the transaction in the SYSADM.CSVTRANS table in the MAIN database. Find the row that corresponds to your transaction ID. The STATE column tells you the transaction status for that record. It can be one of the following:

- IDLE
- PREPARED
- COMMITTING
- ABORTING

If the transaction status is COMMITTING, commit the transaction on the local database. If it is any other state, abort it.

If the commit server itself is damaged and the SYSADM.CSVTRANS table is inaccessible, you need to decide yourself how to resolve the in-doubt transaction. Here are some suggestions:

- Call the administrators at the other sites and find out the state of their transaction. Using the bank situation as an example, if the Dallas branch committed a debit of $500, the Austin branch should go ahead and commit the deposit record.
- Check the data affected by the transaction. For example, if the Dallas branch shows a debit of $500, it may be waiting for a final COMMIT message from the coordinator. You can manually force this commit yourself.

Forcing the resolution

After deciding how to resolve the transaction, use the following commands to force either a commit or rollback:

```
COMMIT TRANSACTION <transaction-id> FORCE
```

or

```
ROLLBACK TRANSACTION <transaction-id> FORCE
```

After you force the resolution, all information about the transaction is deleted from the SYSADM.SYSPARTTRANS, CSVTRANS, and CSVPARTS tables automatically.

Both these commands require DBA privilege.

Performance issues

While two-phase commit is critical to guaranteeing distributed transaction integrity, realize that it does generate network traffic and can affect your database performance.

If there are *X* number of systems involved, the coordinator must send and receive at least 4*X* messages to commit the transaction. This is in addition to the messages that carry the SQL statements and query results. A result of this network overhead can be a decrease in database performance.

In addition, the coordinator must open a second cursor to each database to send the two-phase commit messages.

To improve performance, your database design should maintain frequently accessed data on one system as much as possible. And while a distributed transaction can be the most efficient way to carry out a command, consider the performance issues carefully before using it.

Chapter 10

ODBC Support

This chapter describes SQLBase's support for the Microsoft Open DataBase Connectivity (ODBC) standard. The ODBC standard is an application programming interface (API) specification written by Microsoft. It is an effort to standardize the way in which frontend products access database servers from different vendors.

How does it work?

Currently, each relational database vendor has its own proprietary API. If you want your client application to access a database server, you must code to the database vendor's API.

Alternatively, you can use a router or gateway product to translate from one vendor's API to another vendor's API. Centura's SQLRouter and SQLGateway products provide such a service. They sit between a Centura SQLWindows or Quest client application and a non-SQLBase database server and translate Centura API calls into non-SQLBase API calls. Centura has SQLNetwork products that support DB2, DB2/400, Oracle, SQL Server, Informix, Ingres, and more.

The ODBC standard takes another approach. It calls for all client applications to write to the ODBC standard API and for all database vendors to provide support for it. It then relies on third-party access tools or database drivers (such as the Centura SQLBase and SQLHost/DB2 ODBC Driver) that conform to the ODBC specification to translate the ODBC standard API calls generated by the client application into the database vendor's proprietary API calls.

ODBC overview

The ODBC architecture specifies four components:

- The *application*. The application requests a connection with a data source. It calls ODBC functions to submit SQL statements and retrieve results.
- The *Driver Manager*. The Driver Manager is a dynamic-link library (DLL) that is provided by Microsoft and is included with the Centura SQLBase and SQLHost/DB2 ODBC Driver. Its primary purpose is to load drivers on behalf of an application.
- The *driver*. To access SQLBase, you use the Centura SQLBase and SQLHost/DB2 ODBC Driver, which is packaged with SQLBase.

  The driver processes ODBC function calls, submits SQL requests to a specific data source, and returns results to the application. If necessary, the driver modifies an application's request so that the request conforms to syntax supported by the associated DBMS.
- The *data source*. This consists of the data you want to access and its associated DBMS (in this case, SQLBase), operating system, and network platform (if any).

For more detailed information on ODBC architecture and implementation, read the Microsoft's *Programmer's Reference* manual that accompanies the ODBC Software Developer's Kit (SDK).

SQLBase and SQLHost/DB2 ODBC driver

The following sections describe how to use the Centura SQLBase and SQLHost/DB2 ODBC Driver. The sections discuss:

- System requirements for the Centura SQLBase and SQLHost/DB2 ODBC Driver.
- How to configure data sources.
- How to connect to data sources.
- The mapping between SQLBase data types and the standard ODBC data types.
- Multiple connections and statements.
- Isolation and lock levels.

System components

The Centura SQLBase and SQLHost/DB2 ODBC Driver supports the following Centura components:

- SQLBase version 5.2 or higher.
- SQLHost/DB2 version 3.3.
- SQLGateway/APPC for OS/2 version 3.3.
- SQLGateway/APPC for NetWare version 3.3.
- Windows Client Support version 3.3 (for SQLNetwork for DB2) or version 5.2 or higher (for SQLBase).

  The Centura SQLBase and SQLHost/DB2 ODBC Driver does not support the SQLGateway/DRDA for OS/2. You can not use the Centura SQLBase and SQLHost/DB2 ODBC Driver to access DB2/2.

System requirements

The Centura SQLBase and SQLHost/DB2 ODBC Driver (*gpgup06.dll)* requires:

- SQLBase version 5.2 or higher.
- Microsoft Windows version 3.1 (or higher).
- The Microsoft 2.0 Driver Manager. This is included with the Centura SQLBase and SQLHost/DB2 ODBC Driver.
- The ***sqlapiw.dll*** file and one or more SQLBase communication files to provide access to SQLBase databases. The directory containing these files must be on your path.

If you attempt to configure a data source and you do not have ***sqlapiw.dll*** on your path or in your Windows \system directory, you receive the following error message:

```
The setup routines for the Q+E SQLBase ODBC driver
could not be loaded. You may be low on memory and need
to quit a few applications.
```

Configuring data sources

Use the following procedure to configure a SQLBase data source:

1. Double-click the ODBC icon in the Windows Control Panel to display the data source list.
2. Indicate either a new or existing data source, as follows:
   - To configure a *new* data source, click **Add. A list of installed drivers appears. Select SQLBase and click OK.**
   - To configure an *existing* data source, select the data source name and click **Setup** (in some environments this button is called **Modify**).
3. The Setup box appears. Specify values in this box as follows.

Data Source Name:	Identifies a single connection to a database system. You enter any descriptive string, such as “Accounting” or “SQLBase-Serv1.”
Description:	An optional long description of a data source name. For example, “My Accounting Database” or “SQLBase on Server Number 1.”
Database Name:	For SQLBase, the name of the database system to which you want to connect, or the actual local SQLBase database name. For SQLHost/DB2, the name that is advertised by the gateway on the server.
Server Name:	Optionally, the name of the server that services the desired database. Enter the word LOCAL if you are using the local server.
Server list:	The list of servers that appear in the Logon dialog box. Separate the server names with commas. Enter LOCAL to add the local server.

Translate: (Windows NT and Windows 95 only)	The option to translate your data from one character to another. When selected, this option displays a list of available translators. Choose the Intersolv OEM ANSI translator to translate your data from the IBM PC character set to ANSI.
Default Logon ID:	The default Logon ID used to connect to your database system. Your ODBC application may override this value or you may override this value in the Logon dialog box. A Logon ID is required only if security is enabled on your database system. This ID is case-sensitive.
Cursor Cache Size:	The number of cursors the cursor cache can hold. The default is 6 cursors.
Yield Proc:	This value can be 0 (peek and dispatch), 1 (no yielding), 2 (the Centura SQL/API default yield procedure), or 3 (dispatch via Windows Yield function). It is recommended that you choose the value 1.
Input Message Size:	The number of bytes in the input message buffer. The default is 4000 bytes.
Lock Time Out:	The number of seconds to wait for a lock to be freed before raising an error. Values can be -1 (wait forever) to 1800; the default is 300. This value is not applicable to SQLHost/DB2.
No Recovery:	Selecting this check box disables transaction recovery. Selecting this box is dangerous because your database system can become inconsistent in the event of a system crash. This value is not applicable to SQLHost/DB2.
Release Plan: (Windows NT and Windows 95 only)	A number that determines whether a lock is maintained on a table when the cursors accessing the table are released. When this option is set to 0 (default setting), no locks are freed. To free locks, set this option to 1.

4. Click **OK** to exit the Setup dialog box.

The driver writes these values to the ***odbc.ini*** file. These values are now the defaults when you connect to the data source. You can change the defaults by configuring your data source again. You can insert lines in the appropriate data source section of the ***odbc.ini*** file for any attribute that is not supported in the Setup dialog box or the Logon dialog box.

Connecting to a data source using a logon dialog box

Some ODBC applications display a Logon box when you connect to a data source. “SQLBase” appears in the dialog box title regardless of whether the data source is SQLBase or SQLHost/DB2. Enter the following information in the SQLbase dialog box:

Server Name:	Enter the name of the server containing the SQLBase database tables you want to access, or select the server name from the drop-down list. This list displays the server names you specified in the Setup dialog box. If you want to access a local SQLBase database, enter the word LOCAL.
Database Name:	Enter the name of the database system you want to access, or the name of the local SQLBase database name. For SQLHost/DB2, enter the name that is advertised by the gateway on the server.
User Name:	Enter your user name.
Password:	Enter your password.

Click OK to complete the logon and update the values in the *odbc.ini* file.

Connecting to a data source using a connection string

If your application requires a connection string to connect to a data source, you must specify the data source name that tells the driver which section of the ***odbc.ini*** file to use for the default connection information. Optionally, you may specify attribute or value pairs in the connection string to override the default values stored in the ***odbc.ini*** file.

You can specify either long or short attribute names in the connection string. The connection string has the following format:

```
DSN=data source name[;attribute=value[;attribute=value]...]
```

An example of a connection string for SQLBase is:

```
DSN=SQLBASE_TABLES;SRVR=QESRVR;DB=PAYROLL;
   UID=JOHN;PWD=XYZZY
```

The following table lists the attributes used by SQLBase.

Attribute	Purpose
DataSourceName (DSN)	A string that identifies a single connection to a SQLBase or SQLHost/DB2 database system, for example, “Accounting”.
Database (DB)	The name of the database system to which you want to connect.
ServerName (SRVR)	The name of the server that services the SQLBase or SQLHost/DB2 database tables you want to access. This attribute is optional.
Servers (SRVRLIST)	A list of servers, separated by commas, that display in the Logon dialog box.
LogonID (UID)	The default Logon ID used to connect to your SQLBase system. A Logon ID is required only if security is enabled on your database system. If so, contact your system administrator to get your Logon ID. This value is case-sensitive.
Password (PWD)	Case-sensitive password.
CursorCacheSize (CCS)	The number of cursors in the cursor cache. The cursor cache improves performance and uses few database resources.
InputMessageSize (IMS)	Controls the number of bytes of the input message buffer. Increasing this value retrieves more records across the network in a single fetch.
LockTimeOut (LTO)	Number of seconds SQLBase waits for a lock to be freed before raising an error. Values can be -1 to 1800. A value of -1 is infinite wait. Default is 300.
DB2IsolationLevel (DIL)	Provided for use with SQLHost/DB2 connections to identify the DB2 isolation level being used. Set DIL to CS to use the Cursor Stability isolation level. Set it to RR to use Repeatable Read.
NoRecovery (NR)	Enables or disables transaction recovery. NR=0 enables recover; NR=1 disables recovery. NR=1 improves performance but is dangerous because your database can become inconsistent in the event of a system crash. Read your SQLBase documentation for information on this option.

Attribute	Purpose
YieldProc (YLD)	Determines if you are allowed to work in other applications when the Centura SQL/API is busy. The values are: • YLD=0 (peek and dispatch) — Causes the driver to check the Windows message queue and to send any messages to the appropriate Windows application. • YLD=1 (no yielding) — Does not allow you to work in other applications. • YLD=2 (default yield procedure) — Uses the Centura SQL/API's default yield procedure. • YLD=3 (dispatch via Windows Yield function) — Gives control to the Windows kernel. The Windows kernel checks the message queue and sends any messages to the appropriate application window. It is recommended that you use YLD=1. YLD=0, YLD=2, and YLD=3 do not work with all Windows applications.

Data types

The following table shows how the SQLBase and DB2 data types map to the standard ODBC data types.

SQLBase	DB2	ODBC
char	CHAR	SQL_VARCHAR
varchar	VARCHAR	SQL_VARCHAR
long varchar	LONG VARCHAR	SQL_LONGVARCHAR
date	DATE	SQL_DATE
time	TIME	SQL_TIME
timestamp	TIMESTAMP	SQL_TIMESTAMP
real	REAL/FLOAT	SQL_REAL
double precision	DOUBLE PRECISION	SQL_DOUBLE
number	No equivalent data type	SQL_DOUBLE
decimal	DECIMAL	SQL_DECIMAL

SQLBase	DB2	ODBC
integer	INTEGER	SQL_INTEGER
smallint	SMALLINT	SQL_SMALLIN

Note: The native SQLHost/DB2 data types Graphic, Long Vargraphic, Char() for Bit Data, Varchar() for Bit Data, and Long Varchar for Bit Data are not supported.

Multiple connections and statements

SQLBase supports multiple connections and multiple statements per connection.

Multithread applications for Windows NT and 95

SQLBase and SQLHost/DB2 driver does not allow concurrent access by multiple threads. Applications must ensure that only one thread access the database at all times.

Stored Procedures

SQLBase stored procedures are now supported.

Isolation and lock levels

SQLBase supports isolations levels 1 (read committed) and 3 (serializable). The default is 1.

SQLBase supports page-level locking.

SQLHost/DB2 supports isolation levels 1 (cursor stability) and 3 (repeatable read). The SQLHost/DB2 isolation level cannot be changed by the ODBC driver.

Driver conformance level

The Centura SQLBase and SQLHost/DB2 ODBC Driver is ODBC Level 1 compliant, supporting all ODBC core and Level 1 functions, and includes some support for Level 2 functions. The following table lists the supported functions.

Function	Description
Core Functions	
SQLAllocConnect	Gets a connection handle.
SQLAllocEnv	Gets an environment handle.
SQLAllocStmt	Allocates a statement handle.
SQLBindCol	Assigns storage for a result column and specifies the data type.
SQLBindParam	Assigns storage for a result parameter and specifies the data type.
SQLCancel	Cancels a SQL statement.
SQLColAttributes	Describes the attributes of a column in the result set.
SQLConnect	Connects to a specific driver by data source name, user ID, and password.
SQLDescribeCol	Describes a column in the result set.
SQLDisconnect	Closes a database connection.
SQLError	Returns additional error or status information.
SQLExecDirect	Executes a statement.
SQLExecute	Executes a prepared statement.
SQLFetch	Fetches a row.
SQLFreeConnect	Releases the connection handle.
SQLFreeEnv	Releases the environment handle.
SQLFreeStmt	Ends statement processing and closes the associated cursor, discards pending results, and optionally, frees all resources associated with the statement handle.

Function	Description
SQLGetCursorName	Returns the cursor name associated with a statement handle.
SQLNumResultCols	Returns the number of columns in the result set.
SQLPrepare	Prepares a SQL statement for later execution.
SQLRowCount	Returns the number of rows affected by an INSERT, UPDATE, or DELETE request.
SQLSetCursorName	Specifies a cursor name.
SQLSetParam	Assigns storage for a parameter in a SQL statement.
SQLTransact	Commits or rolls back a transaction.
Level 1 Functions	
SQLColumns	Returns the list of column names in specified tables.
SQLDriverConnect	Connects to a specific driver by connection string or requests that the Driver Manager and driver display connection dialogs for the user.
SQLGetConnectOption	Returns the value of a connection option.
SQLGetData	Returns part or all of one column of one row of a result set (useful for long data values).
SQLGetFunctions	Returns supported driver functions.
SQLGetInfo	Returns information about a specific driver and data source.
SQLGetStmtOption	Returns the value of a statement option.
SQLGetTypeInfo	Returns information about supported data types.
SQLParamData	Returns the storage value assigned to a parameter for which data will be sent at execution time (useful for long data values).
SQLPutData	Send part or all of a data value for a parameter (useful for long data values).

Function	Description
SQLSetConnectOption	Sets a connection option.
SQLSetStmtOption	Sets a statement option.
SQLSpecialColumns	Retrieves information about the optimal set of columns that uniquely identifies a row in a specified table, and the columns that are automatically updated when any value in the row is updated by a transaction.
SQLStatistics	Retrieves statistics about a single table and the list of indexes associated with the table.
SQLTables	Returns the list of table names stored in a specific data source.
Level 2 Functions	
SQLExtendedFetch	Backward and random fetching (SQLBase only).
SQLBrowseConnect	Returns successive levels of connection attributes and valid attribute values. When a value has been specified for each connection attribute, connects to the data source.
SQLColumnPrivileges	Returns a list of columns and associated privileges for the specified table.
SQLDataSources	Gets a list of available data sources.
SQLExtendedFetch	Extends the functionality of SQLFetch: • It returns rowset data (one or more rows), in the form of an array, for each bound column. • It scrolls through the result set according to the setting of a scroll-type argument.
SQLForeignKeys	This function can return: • A list of foreign keys in the specified table (columns in the specified table that refer to primary keys in other tables). • A list of foreign keys in other tables that refer to the primary key in the specified table.

Function	Description
SQLMoreResults	Determines whether there are more result sets available and, if so, initializes processing for the next result set.
SQLNativeSQL	Returns the text of a SQL statement as translated by the driver.
SQLNumParams	Returns the number of parameters in a statement.
SQLPrimaryKeys	Returns the column names that are the primary key for a table.
SQLSetPos	Sets the cursor position in a rowset and allows an application to refresh, update, delete, or add data to the rowset.
SQLSetScrollOptions	Sets options that control the behavior of cursors associated with statement handle.
SQLSetStmtOption	Sets options related to a statement handle; is used with SQLExtendedFetch.
SQLTablePrivileges	Returns a list of tables and the privileges associated with each table

Chapter 11

National Language Support

SQLBase supports English as its standard language, but it also supports many international languages including those spoken in Europe and Asia. This chapter briefly describes how to configure SQLBase to support languages other than English.

NLS configuration files

There are six files that you need to work with in enabling SQLBase to support a language other than English:

- sql.ini
- country.sql
- country.dbs
- *error.sql*
- *country.tlk*
- *message.sql*

sql.ini

In order to support a non-English language, you need to configure a *country* keyword in two sections of the sql.ini file: [sqltalk] and [sqlapiw].

The *country* keyword instructs SQLBase to use the settings in the specified section of the *country.sql* file. For example, if you want your database server to support French, specify:

```
...
[sqltalk]
country=france
[sqlapiw]
country=france
...
```

The matching entry in the *country.sql* file would look like this:

```
[france]
keyword1=value
keyword2=value
```

Read *Chapter 3, Configuration (sql.ini)* for detailed information on the *country* keyword.

country.sql

The *country.sql* file is designed to hold most of the information necessary to customize SQLBase for non-English languages.

This file contains a section for each country that SQLBase supports and each section name is the name of a country.

For example, the *country.sql* file contains the following sections:

```
[france]

[japan]

[germany]
```

The section name corresponding to the country you want SQLBase to support should match the value of the configuration file's (sql.ini) *country* keyword.

Directives

Each section contains statements called directives which specify options and settings. Each directive starts with a pound sign (#) followed by the directive name as well as data that SQLBase can use to perform operations dependent on language differences. A description of each directive follows.

The data that you specify after a directive contains standard characters or character strings which you can edit with a standard text editor. Specify special or non-printable characters in hexadecimal notation: \hex-number. The back-slash (\) character means that a hexadecimal number follows. The number is the hexadecimal (binary) representation of a character.

Code page

Some (but not all) of the directives combine together to make up the code page. The code page is a 256-byte table in memory that describes the attributes of each character.

SQLBase starts out with a default code page that is used for standard English. If you specify any one of the code page directives, SQLBase redefines the *entire* code page. Therefore, if you specify one of the code page directives then you must specify all of the code page directives. The default code page is completely overwritten with new information specified by code page directives.

The following table shows the directives that affect the code page and those that do not.

Affect the code page	Do not affect the code page
alpha firstbyte identifier numeric punctuation secondbyte whitespace	lower prefix translate upper

Sorting

SQLBase uses the Postman Sort algorithm to sort the default English settings. For non-English sorting, SQLBase uses an alternative sorting algorithm to deal with special ASCII/ANSI sort orders. This switch is done automatically when SQLBase detects a national language setting in *country.sql*.

country.dbs and country.tlk

Both files are required to build a database with National Language Support. *country.tlk* is a script that is applied to *country.dbs* to create *start.dbs*. (Read the following section on building a database with National Language Support.)

error.sql

All Centura client and server software need this error file. Besides translating the text of each error message from English to a non-English language, you should also translate the reason and remedy explanations for each error message.

message.sql

All Centura client and server software need this message file too. It contains prompts, confirmations, and other non-error messages for Centura software.

country.sql directives

This section describes, and provides an example of, each directive.

#alpha

Description

This directive specifies the characters that are alphabetic. Alphabetic characters are significant when SQLBase parses identifiers.

Follow this directive with a list of characters.

Example

The country Japan could have this *alpha* directive:

```
[Japan]
#alpha
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

#firstbyte

Description

This directive specifies the characters that are allowed as the first byte of double-byte characters. Only use this directive for countries which use a double-byte character set.

Follow this directive with a list of characters.

Example

The country Japan could have this *firstbyte* directive:

```
[Japan]
#firstbyte
  \81 \82 \83 \84 \85 \86 \87 \88 \89 \8A \8B \8C \8D \8E \8F
\90 \91 \92 \93 \94 \95 \96 \97 \98 \99 \9A \9B \9C \9D \9E \9F
\E0 \E1 \E2 \E3 \E4 \E5 \E6 \E7 \E8 \E9 \EA \EB \EC \ED \EE
\EF \F0 \F1 \F2 \F3 \F4 \F5 \F6 \F7 \F8 \F9 \FA \FB
```

#identifier

Description

This directive specifies additional characters that are allowed in identifiers, such as column names, table names, and index names. These characters are in addition to the standard alpha-numeric characters allowed in identifiers.

Follow this directive with a list of characters.

Example

The country Japan could have this *identifier* directive:

```
[Japan]
#identifier
                                                        \AE \AF
\B0 \B1 \B2 \B3 \B4 \B5 \B6 \B7 \B8 \B9 \BA \BB \BC \BD \BE \BF
\C0 \C1 \C2 \C3 \C4 \C5 \C6 \C7 \C8 \C9 \CA \CB \CC \CD \CE \CF
\D0 \D1 \D2 \D3 \D4 \D5 \D6 \D7 \D8 \D9 \DA \DB \DC \DD \DE \DF
```

#lower

Description

Some countries require special handling when converting a string to lowercase. SQLBase automatically supports standard English lowercase conversion. The *lower* directive alters the lowercase conversion to match the needs of a country.

Example

If the country Glarpia needs to convert "X" to "n" and "Y" to "m" when doing a lowercase operation, the *lower* directive would be:

```
[Glarpia]
#lower
X n
Y m
```

#numeric

Description

This directive specifies the characters that are numeric. Numeric characters are significant when SQLBase parses identifiers.

Follow this directive with a list of characters.

Example

The country Japan could have this *numeric* directive:

```
[Japan]
#alpha
0 1 2 3 4 5 6 7 8 9
```

#prefix

Description

This directive specifies prefixes that SQLBase should ignore when collating data. (A prefix to a string is the front part of the string. For example, if ‘La’ and ‘Le’ are described as prefixes, them ‘La SQL’ and ‘Le SQL’ would collate in the same position.)

Follow this directive with a list of prefixes.

Example

If SQLBase should ignore the prefixes "LA" and "LE" in the country Glarpia, the *prefix* directive would be:

```
[Glarpia]
#prefix
LA
LE
```

#punctuation

Description

This directive specifies punctuation characters.

Follow this directive with a list of characters.

Example

The country Japan could have this *punctuation* directive:

```
[Japan]
#punctuation
   \A1 \A2 \A3 \A4 \A5
```

#secondbyte

Description

This directive specifies characters that are allowed as the second byte of double-byte characters. Only use this directive for countries which use a double-byte character set.

Follow this directive with a list of characters.

Example

The country Japan could have this *secondbyte* directive:

```
[Japan]
#secondbyte
\40 \41 \42 \43 \44 \45 \46 \47 \48 \49 \4A \4B \4C \4D \4E \4F
\50 \51 \52 \53 \54 \55 \56 \57 \58 \59 \5A \5B \5C \5D \5E \5F
\60 \61 \62 \63 \64 \65 \66 \67 \68 \69 \6A \6B \6C \6D \6E \6F
\70 \71 \72 \73 \74 \75 \76 \77 \78 \79 \7A \7B \7C \7D \7E \7F
\80 \81 \82 \83 \84 \85 \86 \87 \88 \89 \8A \8B \8C \8D \8E \8F
\90 \91 \92 \93 \94 \95 \96 \97 \98 \99 \9A \9B \9C \9D \9E \9F
\A0 \A1 \A2 \A3 \A4 \A5 \A6 \A7 \A8 \A9 \AA \AB \AC \AD \AE \AF
\B0 \B1 \B2 \B3 \B4 \B5 \B6 \B7 \B8 \B9 \BA \BB \BC \BD \BE \BF
\C0 \C1 \C2 \C3 \C4 \C5 \C6 \C7 \C8 \C9 \CA \CB \CC \CD \CE \CF
\D0 \D1 \D2 \D3 \D4 \D5 \D6 \D7 \D8 \D9 \DA \DB \DC \DD \DE \DF
\E0 \E1 \E2 \E3 \E4 \E5 \E6 \E7 \E8 \E9 \EA \EB \EC \ED \EE \EF
\F0 \F1 \F2 \F3 \F4 \F5 \F6 \F7 \F8 \F9 \FA \FB \FC \FD \FE \FF
```

#translate

Description

This directive tells SQLBase how to translate characters or strings of characters when sorting character strings. You only need to specify characters which need to be translated.

Example

If the sorting sequence in the country Glarpia requires:

- "N" to be translated to "Z"
- "W " to be translated to "M"
- "EF" to be translated to "F"
- "O" to be translated to "OH"

Then the *translate* directive would be:

```
[Glarpia]
#translate
N Z
W M
EF F
O OH
```

#upper

Description

Some countries require special handling when converting a string to uppercase. SQLBase automatically supports standard English uppercase conversion. The *upper* directive alters the uppercase conversion to match the needs of a country.

Example

If the country Glarpia needs to convert "n" to "X" and "m" to "Y" when doing an uppercase operation, the *upper* directive would be:

```
[Glarpia]
#upper
n X
m Y
```

#whitespace

Description

This directive specifies characters that are white space. A white space character is ignored during parsing.

Follow this directive with a list of characters.

Example

The country Japan would have this *whitespace* directive:

```
[Japan]
#whitespace
\9 \A \D \20
```

Building a database with NLS

In order to use National Language Support, you must build your database from scratch, using country-specific items; therefore you cannot change from one configuration to another without unloading your database and doing the following three steps:

Building the databases

1. Build the *county.sql* file. (As discussed earlier in this chapter.)
2. Make sure that you have copied the *country.dbs* file to a location that will make it a candidate for a database file. For example:

   ```
   copy country.dbs \centura\demo\demo.dbs
   ```

3. Edit the sql.ini file:

   Add the server and client sections that configure the country name. For 32-bit applications:

   ```
     [dbntsrv]

     COUNTRY=GERMANY

     ...

     [win32client]

     COUNTRY=GERMANY

     [sqltalk] (add this section if it does not exist)

     COUNTRY=GERMANY

   ...
   ```

For 16-bit applications:

```
  [dbwservr]
  COUNTRY=GERMANY
  ...
  [winclient]
  COUNTRY=GERMANY
  [sqltalk] (add this section if it does not exist)
  COUNTRY=GERMANY
  [sqlapiw] (add this section if it does not exist)
  COUNTRY=GERMANY
...
```

4. Run the country script file and apply the script to DEMO, by using the following command from the DOS prompt:

   ```
   sqltalk bat db=demo input=country.tlk output=country.out
   ```

5. Make copies of your new newstart.dbs, country.sql, and sql.ini files. Enter the following commands:

   ```
   copy \centura\demo\demo.dbs \centura\newstart.dbs
   copy \centura\demo\country.sql \centura\country.sql
   copy \centura\demo\sql.ini \centura\sql.ini
   ```

6. Make sure you use only one country type at a time from a client without reconfiguring.

Note: Partitioned databases. Make sure you first build the *start.dbs* file with your own *country.sql* file. Then create your partitioned database. Also note that you can’t have a partitioned database with different country settings on the same server: all should have the same country setting used with the MAIN database.

Chapter 12

SQLBase and NetWare Directory Services (NDS)

This chapter describes how to implement SQLBase on the NetWare Directory Services (NDS) 4.x and higher.

The NDS provides global access to all network resources, regardless of their physical location by providing a single, network wide directory. This directory is accessible from multiple points by network users, services and applications.

SQLBase and NDS

Note: This section assumes you are familiar with NDS and how to use it. For details and background on the NetWare Directory Service, read Novell's *NetWare Directory Services Technical Reference Document*.

NetWare 4.x includes the NetWare Directory Services (NDS), which replaces the NetWare 3.x Bindery database. SQLBase Server for NetWare 4.x using the SPX protocol provides support for NetWare Directory Services (NDS). With NDS, you can advertise SQLBase servers and databases on the network in a Directory Tree. You have global instead of local access to all network resources. No longer are you limited to the NetWare Server on which SQLBase resides.

NDS also includes Bindery Emulation, which provides compatibility with bindery-based versions of applications, utilities and services that may coexist with NDS on the network. For details, read *Setting bindery emulation* on page *12-10*.

The SQLBase Server for NetWare 4.x provides a set of SQLBase classes and attributes used to create SQLBase objects in the NDS Tree. This section describes the level of support available for NDS using the SPX protocol, and how to implement SQLBase objects for the Directory Tree.

NDS Support for client and server protocols

NDS support in SQLBase is available for the SPX protocol. All Directory Services clients, except those using Novell's DOS Requester/VLM, are able to see these objects and retrieve relevant information about them. Note that protocols that use NWNETAPI.DLL, such as DOS Requester/VLM, do not provide Directory Service calls.

Following is a matrix that illustrates supported protocols for SQLBase, the corresponding Dynamic Link Libraries (DLLs) for the protocols in each platform, and the flavors of SPX and TCP/IP that are supported in each platform.

Client/Server Platforms

Protocols	**Netware 3.x**	**NetWare 4.x**	**Windows 3.1**	**Windows 95**	**Windows NT**
SPX	spxdll.nlm	spxdll40.nlm tlispx.nlm (Netware 4.1.1)	sqlspxw4.dll	sqlspx32.dll	sqlwsspx.dll
TCP/IP	tlidll.nlm	tlidll.nlm	sqlwsock.dll	sqlws32.dll	sqlws32.dll

Note the following information in this matrix:

- For Windows 95 and Windows NT, the only flavor of TCP/IP that is supported is through Winsock.
- For Windows 95, SPX is supported through Novell's Client32
- For Windows NT, SPX is supported through Winsock.
- For Windows 3.1, the only flavor of TCP/IP is through Winsock. SPX is supported as native SPX
- For NetWare, the only flavor of TCP/IP supported is TLI. SPX is supported as native SPX.

SPX protocol support

SQLBase Server for Netware 4.x/SPX uses NDS in place of Bindery services to store and retrieve information regarding the SQLBase Server. By default, your system is set up for SAP (Service Advertising Protocol) to allow compatibility with the NetWare 3.x Bindery database.

If you are using NDS without Bindery Emulation, you need to configure the keyword *nwadvertisemode* in the [server.spx] section of the sql.ini file. This disables SAP completely and allow objects to be added to the NDS Tree only.

If you plan to use NDS with Bindery Emulation, using SAP advertises the presence of databases and servers on the network. If you have servers and clients relying on SAP location information, you can configure the *nwadvertisemode* to allow both SAP and NDS for advertising on the network.

Since enabling SAP advertising results in additional network traffic, this keyword also lets you turn off servername and installed database names advertising in the network using SAP. Note however, that turning off SAP means that servers and clients relying on SAP will not find the server and the installed databases.

The syntax for this keyword is:

```
nwadvertisemode = 1 | 2 | 3
```

A value of **1** disables NDS completely and enables SAP; a value of **2** disables SAP completely and enables NDS; a value of **3** enables both SAP and NDS.

If you choose to have the Directory Services and the Bindery database co-exist on your platform, you can specify the search order that NDS clients use to connect your SQLBase Servers for NetWare 4.x using the SPX protocol. If a client needs to resolve a server/database name and its mapping to an address, you can specify if the client searches the Directory Services before the Bindery, or vice versa.

To specify the search order, use the optional keyword *preferrednameservice* in the [client.spx] section of sql.ini. For Novell Client32 users, *preferrednameservice* is set to **NDS**, by default, so that the Directory is searched before the Bindery. For non-NDS users, *preferrednameservice* is set to **BIN**, by default, so that the Bindery is searched before the Directory.

Note: The setting for the search order is only valid if the logged in client type is NDS (userid is NDS object). If the logged in client is defined as a Bindery type, then only the Bindery is searched.

TCP/IP protocol support

SQLBase Server for Netware 4.x/TCP/IP uses NDS to store information regarding the server's IP address and installed databases on the SQLBase server. If you are using this platform and want the NetWare 3.x Bindery database to co-exist with NDS, you must specify the keyword *serverpath* in the [client.protocol] section in sql.ini. This keyword is used to resolve a server/database name and its address mapping on the Bindery.

Note: Unlike the SPX protocol, the SAP (Service Advertising Protocol) and the SQLBase keyword *nwadvertisemode* do not apply to TCP/IP. Instead, the TCP/IP *serverpath* keyword (required for using the NetWare 3.x Bindery database) ensures the client is aware of its endpoint address even before the client connects to that endpoint.

Setting up the system for NDS

To implement SQLBase on the Directory Tree, you need to:

1. Determine NDS support available for your network protocol. For details, read *Protocol Support for NDS* in the previous section.

2. If you are using NDS only (without the Bindery database), be sure all clients are upgraded to SQLBase Release 6.2 and use the appropriate comdll.

3. Load DSAPI.NLM which contains the Directory Services functions and is normally already loaded on 4.x file servers.

   To do this, you can either change the file autoexec.ncf, or you can load this NLM along with other NLMs used by SQLBase server.

Note: Step 2 is required whether or not you plan to enable NDS.

4. Run the NLM called "ndsschma.nlm" provided with SQLBase.

   This NLM extends the Directory Services schema by creating class definitions for the SQLBase server and database and provides supervisory rights to add and delete the SQLBase classes that reside at the root of the Directory Tree. You use these classes later on to create SQLBase objects in the NDS Tree.

5. After running ndsschma.nlm, select option **1**, **Add SQLBase Classes and Attributes**, from the main screen of the utility.

   Option 1 creates the class definitions required by SQLBase. When you add SQLBase classes and attributes to the base set of classes that NetWare provides, you are extending the Directory Schema to accommodate SQLBase objects for global access in the network.

6. Repeat steps 3 and 4 for each SQLBase NetWare 4.x server.

7. Create a userid and password with DBA create rights to the part of the Directory Tree where the SQLBase objects reside.

8. Configure each SQLBase NetWare Server 4.x for NDS. For details, read *Configuring SQLBase Server for NetWare 4.x for NDS* on page *12-5*.

9. If applicable to your site, provide optional configuration settings for each NDS client. For details, read *Configuring Clients for NDS* on page *12-7*.

Configuring SQLBase Server for NetWare 4.x for NDS

You can provide optional configuration settings to specify the exact location to add SQLBase objects and the userid/password for logging into the NDS tree. If you are using the SPX protocol and Bindery Emulation, you can also specify if you want to disable the Service Advertising Protocol (SAP). For details on disabling SAP, read *SPX protocol support* on page *12-3*.

Specifying location for SQLBase objects

The *insertioncontext* keyword is optional. It allows you to specify where in the NDS Directory Tree the SQLBase server and database objects are to reside. If you do not specify this keyword, the default context of the user specified by the *ndsloginid* keyword is used for insertion.

The syntax for this keyword is:

INSERTIONCONTEXT= *object_name*

The *object_name* can be a complete or partial object name, containing the following standard abbreviations, which are the most commonly used name types:

Object type	Abbreviation
Country	C
Organization Name	O
Organizational Unit	OU
Name	N
Locality Name	L
Common Name	CN
State or Province Name	S
Street Address	SA

If you use a complete object name you provide the path from the root of the NDS Tree to the particular object. For example, if you want to provide an insertion context for the DEMO database in the SQLBase organizational unit (shown in the Centura NDS tree at the end of this chapter) you enter:

```
[dbnwsrv]
insertioncontext=CN=SERVERxx.OU=SQLBASE.OU=Menlo.O=Centura
```

or, if you want the DEMO database to reside at the root of the NDS Tree, you enter:

```
[dbnwsrv]
insertioncontext=O=CENTURA
```

Note that each path entry is separated by a period. No period is required if you are providing only one path entry, or if you are terminating the path entries. You may also provide a partial object name.

Note: If you are specifying a search context for client users, be sure each *searchcontext* keyword in the client sections of sql.ini is set to the same value as the *insertioncontext* keyword. For details on search context, read *Specifying search context* on page *12-9*.

Specifying user access

The *ndsloginid and ndsloginpassword* keywords are mandatory only if NDS is used for advertising SQLBase servers and databases. Otherwise these keywords are ignored.

These keywords allow you to specify what userid/password are used to log into the NDS Directory Tree for the purposes of adding, deleting, and modifying the SQLBase server and database object entries.

The syntax for this keyword is:

ndsloginid=*userid*
ndsloginpassword=*password*

For example:

```
[dbnwsrv]
ndsloginid=admin
ndsloginpassword=dev
```

Specifying advertising mode

The *nwadvertisemode* keyword is mandatory if NDS is used for advertising SQLBase servers and databases. Otherwise the default for this keyword is SAP only.

This keyword allow you to specify the mode of advertising for SQLBase servers and databases.

The syntax for this keyword is:

```
nwadvertisemode=1 |2 | 3
```

Specify a value of 1 for SAP only (default), 2 for NDS only, and 3 for both SAP and NDS. For example:

```
nwadvertisemode=2
```

Configuring Clients for NDS

Be sure that all clients that intend to log into NDS have an NDS logon ID. When logging into the system, the primary logon ID provided by the client must be one used exclusively for NDS.

You can provide optional configuration settings to determine how SQLBase will search for user objects when an *NDS client* logs into a NetWare Server. Using optional client configuration settings allows you to specify:

- The search order if both NDS and the Bindery database exist on your system using SPX protocol.
- The search context used to search the Directory Tree if NDS resides on your system using the SPX protocol.

For details on protocol support, read *NDS Support for client and server protocols* on page *12-2*. The following sections provide information on keywords and settings to use in the client section of sql.ini.

Client Sections of sql.ini

Depending on the platform you are using, you provide the keywords described in the Client section of sql.ini. The name of the client section for each platform and protocol are listed below.

Client Sections

Protocol	**NetWare 3.x and 4.x**	**Windows 3.1**	**Windows 95 and NT**
SPX	[nwclient.spx] for 3.x [nwclient.spx4]for 4.x	[winclient.spxw4]	[win32client.wsspx] for MS Client for Netware [win32client.spx32] for Novell's Client32 for Netware
TCP/IP	[nwclient.tip] both 3.x and 4.x	[winclient.wsock]	[win32client.ws32]

Note: Currently, Novell's Client32 for NetWare operates on Windows 95 only and MS Client for NetWare operations on Windows NT only.

Specifying search order

The *preferrednameservice* keyword is optional. It is used to specify whether NDS is searched before or after the Bindery database (in the SPX protocol) when an NDS client logs into the server. The default is NDS when an NDS client logs into the server. If the client is non-NDS, the keyword is ignored and only the Bindery database (SPX) is searched.

The syntax for this keyword is:

```
preferrednameservice=NDS | BIN
```

If no keyword is specified in the client section, the default is **BIN** (Bindery) for SPX.

For example, assume you want to specify for a Windows NT SPX client that NDS is searched before the Bindery database. In the Windows NT client section of sql.ini, you specify:

```
[winclient.spxw4]
preferrednameservice=NDS
```

Specifying search context

The *searchcontext* keyword is optional. It is used to specify the part of the NDS Directory Tree that an *NDS client* requires in searching for the SQLBase server name and installed database names.

You can specify more than one searchcontext keyword. If you do specify two or more keywords, the search begins from the first to the last search context that you specify, and if the object is not found, the current context of the logged in user is searched. If the client is non-NDS, the keyword is ignored.

Note: Be sure the *searchcontext* keyword is set to the same value as the *insertioncontext* keyword in the [server] section of sql.ini.

The syntax for this keyword is:

searchcontext= *object_name*

The *object_name* can be a complete or partial object name, containing the standard abbreviations used in the NDS Directory Tree, separated by periods. For examples, see *Specifying location for SQLBase objects* on page *12-5*.

Assume you want to specify that Windows NT SPX clients search a part of the Directory Tree identified as O=CENTURA. You then specify:

```
[winclient.spxw4]
searchcontext=O=CENTURA
```

If no keyword is specified in the client section, the entire NDS Directory Tree is searched.

Administering SQLBase on NDS

NetWare provides the NWAdmin utility so that NetWare Administrators can manage all network resources. To allow the NetWare Administrator to manage SQLBase server and installed databases in the Directory with NWAdmin, SQLBase provides a Windows SNAPIN DLL module. This SQLBase module provides a graphical view of the SQLBase Server and database object instances, including its properties.

The SNAPIN module is currently available for 16-bit Windows. To load the DLL along with the NWAdmin utility, add the following line to the [Snapin Object DLLs] section of NWADMIN.INI:

```
[Snapin Object DLLS]
SNAPIN=CNWA16.DLL
```

Note that the NWADMIN.INI file is located either in the same directory as the executable (SYS:PUBLIC) or in your windows directory.

Be sure also that CNWA16.DLL is copied either to the windows directory or in the path.

Setting bindery emulation

NetWare 4.x includes the Bindery Emulator to provide compatibility with Bindery-based objects from other applications, utilities, and services. The Bindery Emulator retrieves information stored in the Bindery portion of the Directory database on NetWare 4.x and sends Bindery-like information to Bindery clients.

Bindery context is a keyword available on Netware 4.x servers that you can set at the console with the SET command or in the autoexec.ncf file. For example:

SET BINDERY CONTEXT=*object_name*

The *object_name* can be a complete or partial object name, containing the standard abbreviations used in the NDS Directory Tree, separated by periods. For examples, see *Specifying location for SQLBase objects* on page *12-5*.

Full Bindery emulation is established when you provide the Bindery context for a server. When you set this keyword, it enables Bindery clients to log into the NetWare file server, which means Bindery Clients can have the NetWare fileserver be their Preferred Server. If you do not set Bindery Context on the NetWare 4.x fileserver, client workstations cannot log into or get information from the NetWare fileserver’s Bindery.

Full Bindery emulation means that the emulator is in a *dynamic and static* level. This level provides full-scale emulation of static objects as well as accepting request for dynamic objects. All objects at this level are represented as bindery objects when Bindery requests are serviced.

When no Bindery Context is set, the Bindery Emulator is in a *dynamic only* level, which means the emulation only accepts requests for dynamic objects.

For more details on the Bindery Emulator, read Novell’s *Netware Directory Services Technical Reference Document*.

SQLBase NDS schema extension

SQLBase provides a schema extension utility to define the SQLBase-specific NDS *classes* to your NetWare 4.x installation. This utility is provided as a separate NLM called **ndsschma.nlm**.

This NLM creates SQLBase server objects under objects of both Effective and Container type, while database objects are created only under the SQLBase Server. The NDS Base schema is extended by providing the following two classes for SQLBase:

- SQLBASE::DBServer
- SQLBASE::Database

All attributes used in these two classes are part of the base class **server**. The next section presents the information defined for each of the object classes. Details on each of the classes are presented at the end of this section. Read *Information defined for each object class* to clarify the SQLBase class descriptions described at the end of this section. You can also read Novell's *NetWare Directory Services Schema Specification* part of the Software Developer's Kit for more details.

SQLBase object classes

This section first presents general information defined for each of the NetWare object classes before describing the SQLBase object classes that you add to the base class **server**. Read this section first to clarify the SQLBase class descriptions that follow. You can also read Novell's *NetWare Directory Services Schema Specification* part of the Software Developer's Kit for more details.

Information defined for NetWare object classes

Each NetWare object class has the following information defined:

Object Class Information	Description
SuperClasses	Objects of this class inherit information types and attributes from classes listed here. Top, Server, and Resource are classes available in the Base Schema.
Containment	Objects of this class may only be created as subordinates in the Directory Tree to objects of the classes listed here. An object of this class may not be subordinate to any object of a class that is not listed here.
Named By	The partial name or Relative Distinguished Name (RDN) of objects of the class consists of at least one of the attributes listed here. At least one of the attributes listed here must be given a value when creating an object of the class. Attributes listed in this section are also listed in the “Mandatory Attributes” or “Optional Attributes” sections.

Object Class Information	Description
Mandatory and Optional Attributes	All attributes are either mandatory or optional. If an attribute is mandatory, a value must be assigned to that attribute. If an attribute is optional, an assigned value is not required unless it is the only Named By Attribute.
Default ACL Template	Every expanded class definition has an ACL attribute (inherited from the class Top). This attribute holds information about which trustees have access to the object itself and which trustees have access to the attributes for the object. Each class can have its own default set of values for their ACL but can also inherit ACL values from the super classes.

SQLBASE::DBServer

This class is defined as both an effective and container class. This class belongs to the SuperClass **server** and includes all classes of both Effective and Container type as its containable classes. This means that an object of this type can be created under any object which belongs to classes of both effective and container type.

The Named By attribute is **CN** and the mandatory attribute is **Host Device**. There are no optional attributes. All the attributes are inherited from the super class server of which only the **Network Address** attribute is used.

Super Classes: Server

Containment Classes:*Top
*Country
*Locality
*Organization
*Organizational Unit

Named By:*CN (Common Name)

Mandatory Attributes: Host Device

Optional Attributes:*Network Address

Default ACL Template:

Object NameDefault RightsAffected Attributes
*[Creator]Supervisor[Entry Rights]

Note: Every attribute marked with an * is an inherited attribute from one of its super classes. The SQLBase Server and SQLBase database classes have SQLBASE:: as prefix because it is registered as the requested name prefix for the SQLBase product with Novell's Directory Schema Registry.

SQLBASE::Database

This class is defined as an effective class. This class belongs to the SuperClass **server** and includes SQLBASE::DBServer as the only containable class. This means that an object of this type can be created under objects which belong to class SQLBASE::DBServer.

There are no new Named By, Mandatory and Optional attributes. All the attributes are inherited from the super class **server** of which only the **Network Address** attribute is used.

Super Classes: Server

Containment Classes:SQLBASE::DBServer

Named By: None

Mandatory Attributes: None

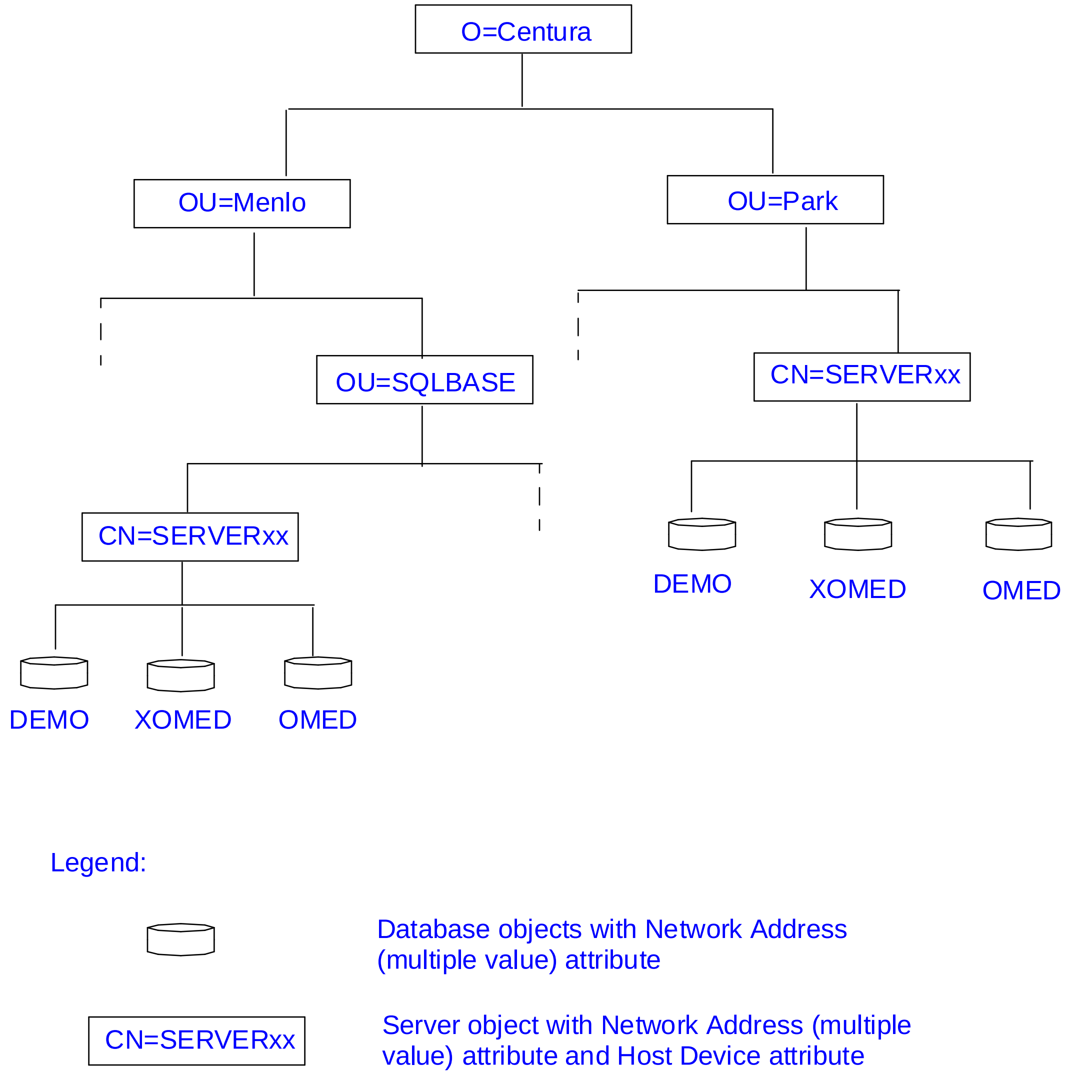
Optional Attributes:*Network Address

Default ACL Template:

Object NameDefault RightsAffected Attributes
*[Creator]Supervisor[Entry Rights]

Note: Every attribute marked with an * is an inherited attribute from one of its super classes. The SQLBase Server and SQLBase database classes have SQLBASE:: as prefix because it is registered as the requested name prefix for the SQLBase product with Novell's Directory Schema Registry.

SQLBase Schema Extension Diagram



Database objects with Network Address (multiple value) attribute

Server object with Network Address (multiple value) attribute and Host Device attribute

Chapter 13

Running SQLBase Server as an NT Service

This chapter contains information on running SQLBase Server as an application program or a Windows NT service program, and using SQLBase Server Monitor. Topics include:

- Overview of service programs
- Installing SQLBase Server for Windows NT as a service program
- Administering service programs
- Using SQLBase Server Monitor
- Services window of the Control Panel
- Using the Windows NT registry
- NT Event Log
- Shutting down SQLBase Server for Windows NT when running as a service program

About service programs

Windows NT programs can be either application programs or service programs. Previously, SQLBase Server for Windows NT ran as an application program, requiring the user to log onto Windows NT before starting SQLBase. This results in SQLBase terminating when the user logs off.

Windows NT service programs run independently of the user, and are unaffected by users logging on and off the NT machine. They do not require the user to log onto the NT machine prior to starting, and can start either automatically or on demand. If started automatically, they begin their execution during Windows NT initialization.

Running SQLBase Server as a Windows NT service program offers the following advantages:

- A user need not be logged onto the system for SQLBase to start.
- Security is enhanced, preventing unauthorized tampering or termination.

You can start SQLBase as an NT Service manually or automatically. When you specify manual start up, you must start Windows NT before the SQLBase Service program starts up. When you specify automatic start up, the SQLBase Service program starts up automatically when NT starts up.

You can administer the SQLBase NT service program using NT Service Manager or SQLBase Server Monitor (SSM).

Installing SQLBase Server for Windows NT

In order to install SQLBase Server for Windows NT as a service program, the user performing the installation must have Administrator privileges.

Running SQLBase Server for Windows NT as a service program is supported on the Windows NT 3.51/4.0 operating system platforms, including the server and workstation versions of Windows NT.

SQLBase Server for Windows NT always installs as a service program. However, it can also be run as an application program. Read the section Administering service programs for details.

The default startup setting is with SQLBase disabled as a service. During installation, SQLBase is set to run as a system account and to interact with the desktop.

Only one instance of SQLBase Server for Windows NT for Win32 can be run on an NT machine at any given time, regardless of whether it is being run as a service program or an application program.

Administering service programs

You can administer SQLBase as an NT service program using one or both of the following tools:

- Centura's SQLBase Server Monitor (SSM) that is part of the SQLBase Server package
- Services window of the Control Panel

Note: SQLBase Server Monitor provides authorization to any user to administer SQLBase Server if they have access to the system running SQLBase. If you want to restrict access to SQLBase Server to a single authorized user or administrator, you must administer SQLBase as an NT service exclusively through the NT Services window.

SQLBase Server Monitor (SSM)

Overview

You access SQLBase Monitor through the Centura Program Group in the Start menu.

Note: You cannot start SQLBase as an application program using SSM. However, you can monitor SQLBase as an application program and stop it from within SQLBase Server Monitor.

SQLBase Server Monitor (SSM) allows you to obtain the status of SQLBase Server while it is running, providing you with a live report of all active SQLBase databases. Using SQLBase Server Monitor, you can automatically monitor your databases and display the following information for each database:

- Location in the form of a fully qualified path to the database
- File sizes
- Free disk space on the database volume for non-partitioned databases
- Free disk space on the log disk volume for non-partitioned databases
- Network protocols in use for sending information to and from the database
- Users currently connected to the database

SSM uses a GUI interface to display information in an easy-to-view tree format. A toolbar lets you start and stop SQLBase as an NT service and update NT registry settings. These tasks are also available in the NT Service Manager.

SSM displays the following categories of SQLBase Server information in a tree structure:

- Listens node

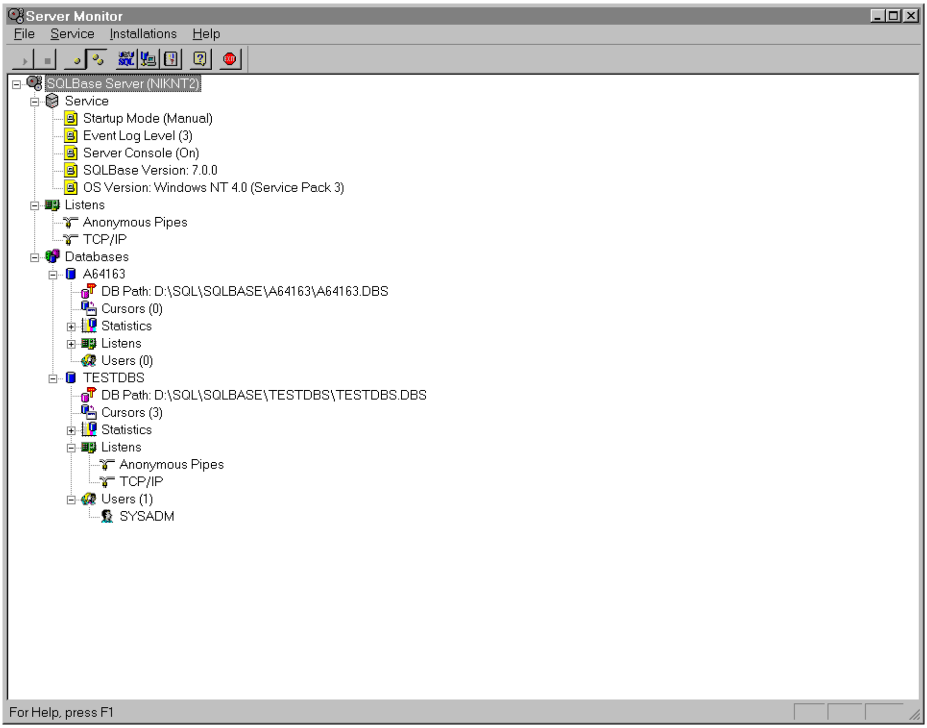
  Displays network protocols in use by SQLBase. Network protocols in use by specific databases are also provided in the Databases node of the tree.

- Databases node

  Displays details and statistics about each SQLBase database. If SQLBase Server is not running when you start up SSM, the interface displays the Service node with the words "Not Running" beside it. All tree information is disabled

- Service node

  Displays the current settings for SQLBase start up behavior, level of error reporting, GUI user visibility, and platform identification.



```
SQLBase Server (NIKNT2)
    Service
        Startup Mode (Manual)
        Event Log Level (3)
        Server Console (On)
        SQLBase Version: 7.0.0
        OS Version: Windows NT 4.0 (Service Pack 3)
    Listens
        Anonymous Pipes
        TCP/IP
    Databases
        A64163
            DB Path: D:\SQL\SQLBASE\A64163\A64163.DBS
            Cursors (0)
            Statistics
            Listens
            Users (0)
        TESTDBS
            DB Path: D:\SQL\SQLBASE\TESTDBS\TESTDBS.DBS
            Cursors (3)
            Statistics
            Listens
                Anonymous Pipes
                TCP/IP
            Users (1)
                SYSADM
```

SQLBase Server Monitor Main Window

Service node

You administer SQLBase Servers in SQLBase Server Monitor using the Service menu or using the Service node. The Service node displays in an easy-to-view tree format in SSM’s main window.

Note: Any changes you make to server settings are not effective until the next time you start SQLBase.

The following options appear in the Services node:

Startup Mode

The Startup Mode determines how SQLBase starts. Access the Startup Mode using the **Service** menu or by clicking **SQLBase Server** node, **Service** node, then **Startup Mode**. Valid options are:

- Manual

  Allows you to start SQLBase as a service (after starting NT) using SSM or the NT Service Manager.

- Automatic

  Allows you to start SQLBase as a service automatically when Windows NT starts up.

- Disabled

  Allows you to start SQLBase as an application only.

Event Log Level

Event Log Level determines the level of events reporting, such as errors, warnings, and messages, from SQLBase to the Event Log. When SQLBase Server for Windows is running as an NT service program, operational and error message are logged to the NT Application Event Log. SQLBase displays no error dialogs when running as a service program. You can view logging events through the SSM Event Viewer, which displays informational, warning, and error events. A parameter in the Windows NT registry controls event logging. Message types include:

- Startup
- Shutdown
- Database(s) being serviced
- SQLBase.INI file error
- Internal logic errors
- Deadlock messages

Through the Event Viewer, you can click on an event to display the detail dialog for the event.

Access Event Log Level using the **Service** menu or by clicking the **SQLBase Server** node, **Service** node, **Event Logging**. Valid event log level settings are: 1, 2, and 3.

Server Console

Server Console determines whether the SQLBase GUI and icon are visible to users. Access Server Console settings using the **Service** menu or by clicking the **SQLBase Server** node, **Service** node, **Server Console**.

Valid Server Console settings are:

- Off

  The SQLBase GUI and icon are not visible to users.

- On

  The SQLBase GUI and icon is visible and enabled for user interaction.

  If SQLBase is running as a service program, its shut down menu item is disabled to prohibit users from shutting down SQLBase.

Platform and version information

SQLBase Server Monitor displays both the SQLBase version and Operating System version.

Listens node

The Listens Node displays the network protocols that SQLBase Server is using.

Databases node

The SQLBase Server Monitor Databases node displays all of the SQLBase databases in the system. SSM displays the following information when you click on a database:

- DB Path

  Operating System path to the database.

- Cursors

  Number of cursors connected to the database.

- Statistics

  Database file size, free space on the database volume, and free space on the log volume.

- Listens

  Network protocols that the database is using (these protocols can be a subset of the protocols SQLBase is using).

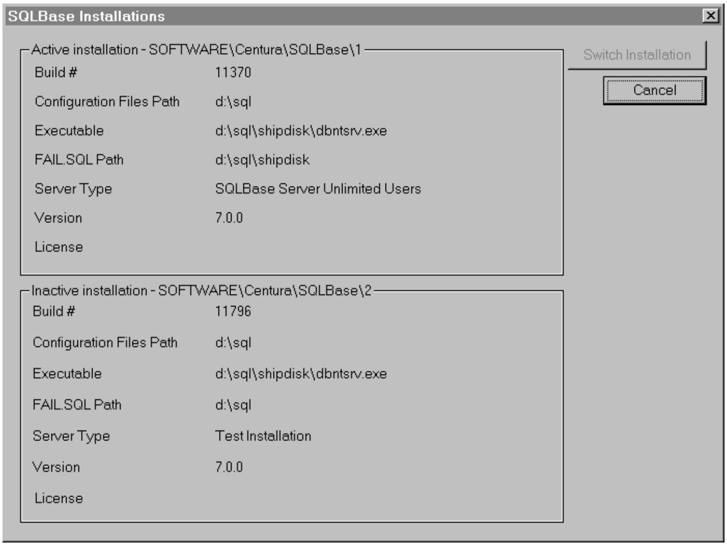
- Users

  Number of users connected to the database and user names. You have the option of disconnecting a user and performing a rollback on any outstanding transactions for that user, by right clicking on the user name.

Displaying server installations

You can display server installation information using the SQLBase Installations window. To display, select **Display Installations** from the **Installations** menu. SSM displays the following information for each installation:

- Installation Name with Active or Inactive specification
- Build Number
- Configuration Files Path
- Executable
- FAIL SQL Path
- Server type
- Version Number
- License



Active installation - SOFTWARE\Centura\SQLBase\1

Build #	11370
Configuration Files Path	d:\sql
Executable	d:\sql\shipdisk\dbntsrv.exe
FAIL.SQL Path	d:\sql\shipdisk
Server Type	SQLBase Server Unlimited Users
Version	7.0.0
License	

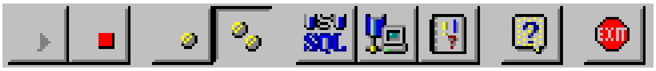
Inactive installation - SOFTWARE\Centura\SQLBase\2

Build #	11796
Configuration Files Path	d:\sql
Executable	d:\sql\shipdisk\dbntsrv.exe
FAIL.SQL Path	d:\sql
Server Type	Test Installation
Version	7.0.0
License	

SQLBase Installations window

Using the SQLBase Server Monitor toolbar

The SQLBase Server Monitor toolbar contains buttons you can click to perform various tasks, such as starting and stopping the SQLBase Server.



SQLBase Server Monitor toolbar

Button	Name	Description
▶	**Start SQLBase**	This option is disabled if SQLBase is already running. **Note:** You cannot start SQLBase as an application program from within SSM.
■	**Stop SQLBase**	If SQLBase is running as a service or an application program, click this icon to stop SQLBase. A warning message displays if users are currently connected. When you shut down SQLBase as a service using SSM, the shut down is unconditional. For more information, read *Shutting down SQLBase Server for Windows NT* on page *13-15*. When you shut down SQLBase as an application program, shutdown occurs when no users are left. SQLBase will not accept any more connections during this time.
●	**Refresh Display**	Updates the SQLBase Server Monitor display with the current server status.
●●	**Auto Refresh**	Periodically updates SQLBase Server Monitor display to include current server status. Set the refresh rate using **Auto Refresh Settings** in the **File** menu.
SQL	**SQLTalk**	Starts SQLTalk.

	Configuration Utility	Starts the SQLBase Configuration utility.
	Event Viewer	Starts the SQLBase Event Viewer.
	About SQLBase Server Monitor	Displays program information, version number, and copyright.
	Close Server Monitor Window	Closes the Server Monitor Window. To shutdown SQLBase Server Monitor use the task bar.

Troubleshooting SQLBase Server Monitor

To troubleshoot SQLBase Server Monitor, check that the registry settings automatically installed for SQLBase are valid. For a list of SQLBase registry settings, read *Using the Windows NT registry* on page *13-11*.

Warning: Changing the SQLBase Server Monitor settings to values other than those provided by the installation program can adversely affect the operation of SQLBase.

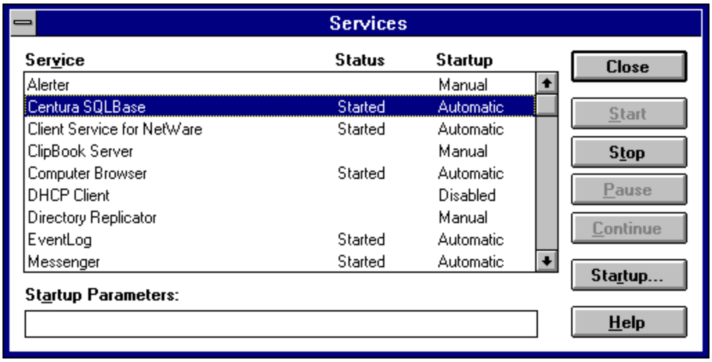
Security

Unlike the NT Service Manager which restricts access to the SQLBase Server to a single authorized user or administrator, SSM provides authorization to any user with access to the SQLBase Server. If you are using SSM, you need to guard against unauthorized use of SQLBase Server.

If SQLBase is run in an account other than the system account, the SQLBase interface is invisible to users and administrative access due to NT security, even if the Console is enabled. Access to SQLBase is gained through SQLBase Server Monitor.

Using the Services window of the Control Panel

You can administer SQLBase as an NT Service program using the Services window of the Control Panel. The Services window displays the current status and startup type of each service program installed on the machine. Use the Start and Stop buttons to start and stop the selected service program.



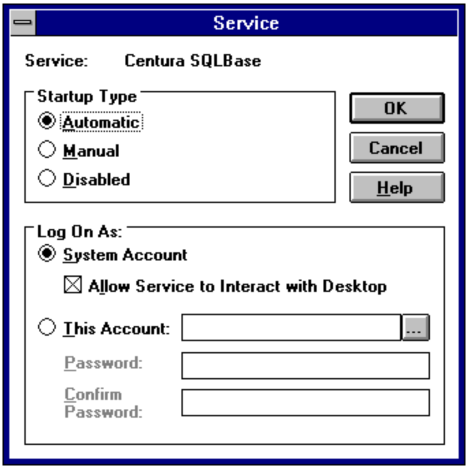
Services

Service	Status	Startup
Alerter		Manual
Centura SQLBase	Started	Automatic
Client Service for NetWare	Started	Automatic
ClipBook Server		Manual
Computer Browser	Started	Automatic
DHCP Client		Disabled
Directory Replicator		Manual
EventLog	Started	Automatic
Messenger	Started	Automatic

Startup Parameters:

Windows NT Services window

The Startup button displays the Service dialog.



Service

Service: Centura SQLBase

Startup Type
● Automatic
○ Manual
○ Disabled

Log On As:
● System Account
☒ Allow Service to Interact with Desktop
○ This Account:
Password:
Confirm Password:

Service dialog

Use this window to modify the properties of the service program, including startup type and the account under which the service program executes. If SQLBase Server for Windows NT is running as a service program, and is running in an account other than the system account, the normal SQLBase interface will be completely hidden. This limitation is imposed by Windows NT. In this situation, administrative access, beyond starting and stopping SQLBase, is gained through SSM. If the service program logs on as a System Account, it can be set to interact with the desktop, but must be stopped using SSM or NT Service Manager.

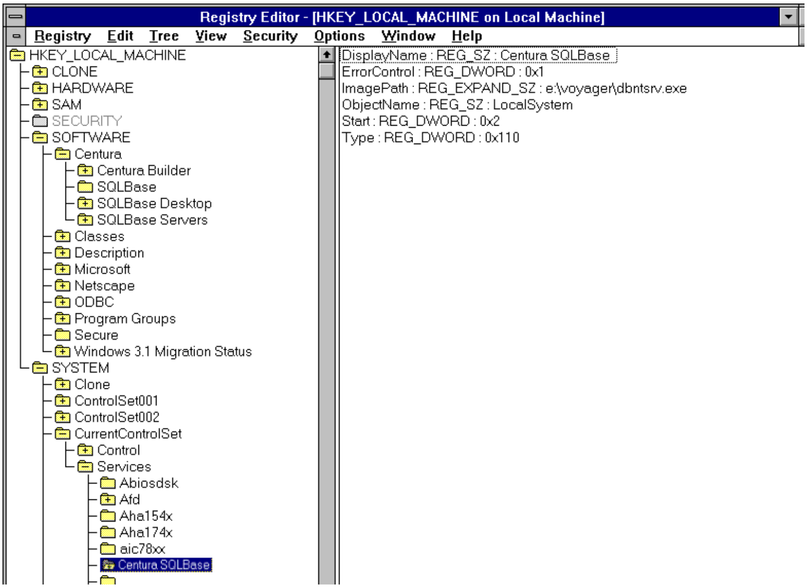
Using the Windows NT registry

All Windows NT service programs have an entry in the Windows NT registry containing parameters that identify for the NT Service Control Manager and SQLBase Server Monitor basic operational parameters relevant to starting the service program. The registry key for this entry is:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\<service-name>
```

The registry key is constructed during SQLBase installation. For SQLBase Server, the service name is Centura SQLBase and the registry key for the service is:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Centura SQLBase
```

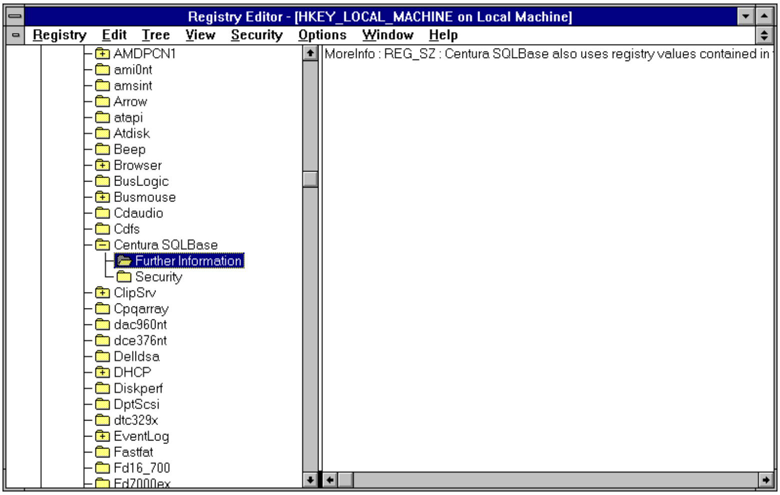


```
DisplayName : REG_SZ : Centura SQLBase
ErrorControl : REG_DWORD : 0x1
ImagePath : REG_EXPAND_SZ : e:\voyager\dbntsrv.exe
ObjectName : REG_SZ : LocalSystem
Start : REG_DWORD : 0x2
Type : REG_DWORD : 0x110
```

Windows services registry key for SQLBase

SQLBase Server, when installed as a service program on NT, also creates the following registry key, providing a cross-reference to the operational parameters location:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Centura SQLBase\
  Further Information
```



MoreInfo : REG_SZ : Centura SQLBase also uses registry values contained in

Windows services registry key

SQLBase operational parameters registry key

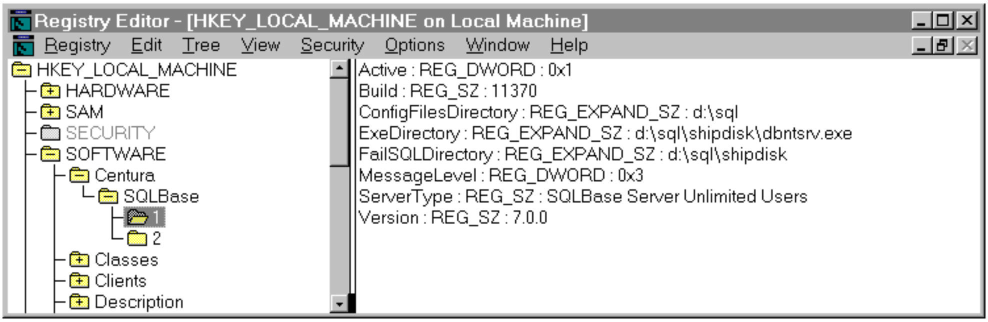
When SQLBase Server (Windows NT and Windows 95) is installed, an entry is put in the Windows registry. The operational parameters that SQLBase obtains from the Windows registry includes:

- The location of the configuration files to use for initialization. These files are:
  - SQL.INI
  - MAIN.INI
  - ERROR.SQL
  - MESSAGE.SQL
  - COUNTRY.SQL
- The event level for messages stored by the NT Event logging facility. (Windows NT only)

The registry key for SQLBase to obtain its operational parameters, such as the sql.ini file location, is in a location separate from its services registry location. The common registry locations for SQLBase to retrieve its operational parameters, whether it is running as a service program or as an application program, are:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Centura\SQLBase\1
HKEY_LOCAL_MACHINE\SOFTWARE\Centura\SQLBase\2
```

Note: You can have more than one installed version of SQLBase, although only one can be active at any given time. Use SQLBase Server Monitor to specify which installation of SQLBase you want to use.



```
Active : REG_DWORD : 0x1
Build : REG_SZ : 11370
ConfigFilesDirectory : REG_EXPAND_SZ : d:\sql
ExeDirectory : REG_EXPAND_SZ : d:\sql\shipdisk\dbntsrv.exe
FailSQLDirectory : REG_EXPAND_SZ : d:\sql\shipdisk
MessageLevel : REG_DWORD : 0x3
ServerType : REG_SZ : SQLBase Server Unlimited Users
Version : REG_SZ : 7.0.0
```

Windows services registry hierarchy key

NT Application Event Log

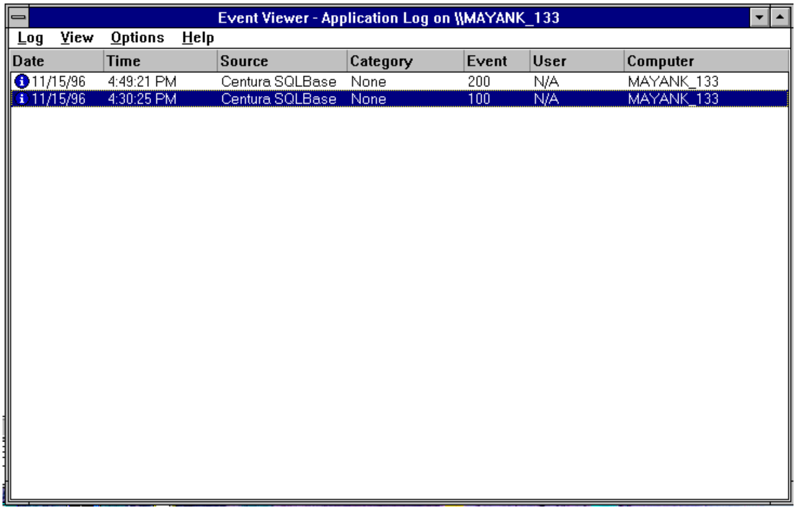
When running SQLBase Server for Windows NT as a service program, operational and error messages are logged to the NT Application Event Log. Message types include:

- Startup
- Shutdown
- Database(s) being serviced
- sql.ini file errors
- Internal logic errors
- Deadlock messages

Note: SQLBase displays no error dialogs when running as a service program, even if configured to interact with the desktop.

The NT Application Event Log supports three types of logging events:

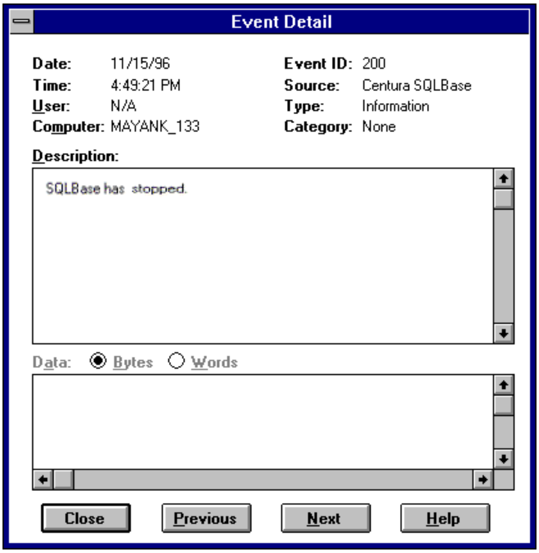
- Informational events
- Warning events
- Error events



Date	Time	Source	Category	Event	User	Computer
11/15/96	4:49:21 PM	Centura SQLBase	None	200	N/A	MAYANK_133
11/15/96	4:30:25 PM	Centura SQLBase	None	100	N/A	MAYANK_133

NT Event Log

Click on an event to bring up the **Event Detail** dialog.



Date: 11/15/96
Time: 4:49:21 PM
User: N/A
Computer: MAYANK_133
Event ID: 200
Source: Centura SQLBase
Type: Information
Category: None

Description:

SQLBase has stopped.

Data: ● Bytes ○ Words

Event Detail dialog message text

A parameter in the Windows NT registry controls event logging. The following table lists the SQLBase level, event type, and message events:

SQLBase Level	NT Event Type	SQLBase Message Events
1	• Informational • Error • Error • Warning • Warning	• Startup and Shutdown • Startup errors (e.g., sql.ini errors) • Database shutdown due to error conditions • Deadlock detection • Shutting down with users still connected

Each SQLBase message event has corresponding message text that displays in the SQLBase GUI when the event occurs. The same text is recorded in the NT Event Log.

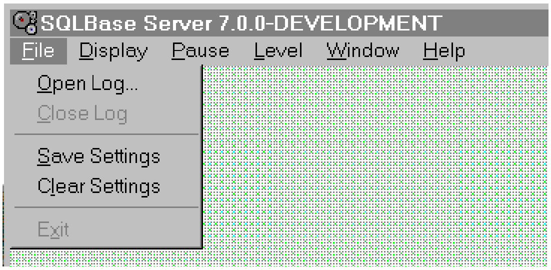
You can also view errors using the Event Viewer in SQLBase Server Monitor. For more information, read *Event Log Level* on page *13-5*.

Shutting down SQLBase Server for Windows NT

When you are running SQLBase Server for Windows NT as an application program, SQLBase can be shut down using SQLBase Server Monitor, the SQLBase File menu, or the SQLBase icon when the user interface is minimized.

When you are running SQLBase Server for Windows NT as a service program, the ability to shut down SQLBase from a menu item is disabled. You can only shut down SQLBase using SQLBase Server Monitor or from the Services window of the Control Panel.

If users are connected to SQLBase, a database recovery is required on the next start up. To prevent data loss, be sure all users are logged off SQLBase before shutting it down. For information on shutting down SQLBase Server using SQLBase Server Monitor, read *SQLBase Server Monitor (SSM)* on page *13-3*.



Disabled File menu shutdown point

Running SQLBase as an application program

SQLBase Server for Windows NT may still be set up to run as an application program, even if it is installed as a service program. You can only run one instance of SQLBase on a machine. If SQLBase is already running as a service, then trying to run SQLBase as an application program will be unsuccessful and result in an error message from the new server. When choosing to run SQLBase as an application program, you must establish an NT session before you start the SQLBase Server. To start SQLBase as an application program, double-click on the SQLBase Server icon rather than starting SQLBase with the NT Service Manager or the SQLBase Server Monitor.

You can shut down SQLBase using SQLBase Server Monitor, SQLBase File menu, or the SQLBase icon when the user interface is minimized. If the SQLBase Server is running when you end an NT session, the SQLBase Server is terminated. When running SQLBase Server as an application program, you can use the SQLBase Server Monitor to report the status of the SQLBase databases.

Appendix A

System Catalog Tables

This Appendix contains descriptions of the tables in SQLBase's system catalog. The tables are organized alphabetically by name.

System catalog summary

The system catalog is a set of tables that contain information about objects in the database. The tables are owned by SYSADM and maintained by SQLBase. The system catalog is also called the *data dictionary*.

Table Name	Description
SYSCOLAUTH	Lists each user's column update privileges.
SYSCOLUMNS	Lists each column of every table.
SYSCOMMANDS	Lists all stored commands and procedures.
SYSDEPENDENCIES	Lists each dependency between a stored procedure and an external function.
SYSEVENTS	Lists all the system timer events.
SYSEXECUTEAUTH	Lists the execution authority levels of users with privileges on each stored procedure.
SYSEXTFUN	Lists all declared external functions.
SYSEXTPARAM	Lists each parameter of an external function.
SYSFKCONSTRAINTS	Lists each foreign key constraint.
SYSINDEXES	Lists each table’s indexes.
SYSOBAUTH	Lists each user who is granted execute privilege on an external function.
SYSOBJSYN	Lists each synonym created for an external function, stored command, or stored procedure.
SYSKEYS	Lists each column in every index.
SYSPARTTRANS	Lists each in-doubt distributed transaction.
SYSPKCONSTRAINTS	Lists each primary key constraint.
SYSROWIDLISTS	Lists information about saved Result sets.
SYSSYNONYMS	Lists all table and view synonyms.
SYSTABAUTH	Lists each user's table privileges.
SYSTABCONSTRAINTS	Lists each table constraint.
SYSTABLES	Lists each table or view.

Table Name	Description
SYSTRGCOLS	Lists each column on which an UPDATE trigger exists.
SYSTRIGGERS	Lists each trigger.
SYSUSERAUTH	Lists each user's database authority level.
SYSVIEWS	Lists the text of each view.

SYSADM.SYSCOLAUTH

This table contains the update privileges of users for individual columns of a table or view.

Column Name	Description
GRANTEE	The authorization ID of the user who holds update privileges.
CREATOR	The authorization ID of the user who created the table on which the update privileges are held.
TNAME	The name of the table or view on which privileges are held.
COLNAME	The name of the column to which the UPDATE privilege applies.

SYSADM.SYSCOLUMNS

This table contains one row for every column of each table and view (including the columns of the system catalog tables).

Column Name	Description
NAME	The name of the column.
TBNAME	The name of the table or view that contains this column.
TBCREATOR	The authorization ID of the user who created the table or view in which the column exists.
COLNO	The relative column number in the table. This number remains the same even after a column is dropped.
COLTYPE	The data type of the column.

Column Name	Description
LENGTH	The length of the data in the column, or for a DECIMAL column, the precision. The standard lengths for the data types are: INTEGER 4 SMALLINT 2 FLOAT 8 CHAR Length of string VARCHAR Maximum length of string DECIMAL Precision of number
SCALE	The scale for a DECIMAL data type column. This is zero (0) for other types.
NULLS	'Y' if nulls are allowed in the column; 'N' if defined with NOT NULL; 'D' if defined with NOT NULL WITH DEFAULT.
UPDATES	'Y' if the column can be updated; 'N' if the column is read-only.
REMARKS	A user-specified comment about each column. The maximum length of a comment is 254 characters.
LABEL	A user-specified label about each row. The maximum length of a label is 30 characters.
AVGCOLLEN	The average column length across all rows in the table. This can differ from the defined column length because SQLBase stores all columns as variable data.
AGCOLLONGLEN	The average length of a LONGVARCHAR column across all rows of the table. This is zero for a non-LONGVARCHAR column.

SYSADM.SYSCOMMANDS

This table contains one row for every stored command or procedure.

Column Name	Description
CREATOR	The authorization-id of the creator of the stored command or procedure.
NAME	The name of the stored command or procedure. When SQLBase stores a command contained in a static procedure, SQLBase assigns it an internally-generated name.
TYPE	Command (C) or procedure (P).
SYSTEM	Contains ’Y’ if SQLBase created the stored command or procedure; ’N’ if a user created it. When SQLBase stores a command or procedure contained in a static procedure, this value is 'Y'. Conversely, when you explicitly STORE a command or procedure, this value is 'N'.
STATIC	'Y' if the stored procedure is static; ‘N’ if it is not.
VALID	Contains ’Y’ if the command is valid, and ’N’ if the command is invalid. A stored command can become invalid in a situation such as when you delete a referenced column, or drop an index on a referenced column. Stored procedures do not become invalid.
AUTORECOMPILE	If set to ON and the stored command later becomes invalid, SQLBase automatically recompiles the stored command the next time you execute it with either the SQLTalk EXECUTE command or the SQL/API *sqlret* function.
TEXT	The text of the stored command or SQL statements in the procedure.
SNUM	An integer column. This is an identification (serial number) associated with SYSCOMMANDS.

SYSADM.SYSDEPENDENCIES

This table currently contains one row for each dependency between a stored procedure and an external function.

Column Name	Description
DETCREATOR	The authorization ID of the user who created the determinant object, which is the external function.
DETNAME	The name of the determinant object, which is the external function. This is the name assigned to the function when it is created.
DETTYPE	The type of determinant object. ‘F’ for function.
DEPCREATOR	The authorization ID of the user who created the dependent object, which is the stored procedure.
DEPNAME	The stored procedure name which is the dependent object of the external function.
DEPTYPE	The type of dependent object. ‘P’ for procedure.

SYSADM.SYSEVENTS

This table contains one row for every event.

Column Name	Description
CREATOR	The authorization ID of the creator of the event.
NAME	The name of the event.
TYPE	Command (C) or procedure (P).
TYPEID	The event type. Currently, only the timer event type is supported.
EVENTID	The ID of the event.
BEGINTIME	The start time of the event.
INTERVAL	The time interval (in seconds) between executions of the event.
SYSTEM	'Y' if the event is system-defined; ‘N’ if it is not.
SPSNUM	An integer column that refers to the serial number (SNUM in SYSCOMMANDS) of the stored procedure that is executed by an event.

Column Name	Description
TEXT	The text of the stored command or SQL statements in the procedure.
ACTIONCODE	The compiled code of the stored command or SQL statements in the procedure. This is for internal use only.

SYSADM.SYSEXECUTEAUTH

This table contains one row for every GRANT PRIVLEGES command run for stored procedures.

Column Name	Description
CREATOR	The authorization ID of the user who created the stored procedure
NAME	An internally-generated name for the stored procedure.
GRANTEE	The authorization ID of the grantee (user of the stored procedure).
USECREATORPRIV	’Y’ if the GRANTEE is using the creator’s privileges; blank otherwise.
USEGRANTEEPRIV	’Y’ if the GRANTEE is using his own privileges; blank otherwise.

SYSADM.SYSEXTFUN

This table contains one row for each declared external function.

Column Name	Description
CREATOR	The authorization ID of the user who created the external function.
NAME	The name of the external function.
EXTNAME	The external name of the external function.
LIBNAME	Full path name of the library where the external function resides
SNUM	Unique SQLBase ID for the external function.
NUMPARAMS	Number of parameters in the external function.

Column Name	Description
RETURN	’Y’ if the external function returns a value; ’N’ otherwise.
EXECMODE	Function execution mode: ‘S’ for same thread, or ‘P’ for separate process.
CALLSTYLE	Callstyle for the external function. It can be one of the following: • CDECL • STDCALL • PASCAL
REMARKS	User-specified comment
LABEL	User-specified label.

SYSADM.SYSEXTPARAM

This table contains one row for each parameter of an external function.

Column Name	Description
FUNSNUM	Unique SQLBase ID for the external function.
POSITION	Position of the parameter as defined in the function (1, 2 ...).
EXTTYP	External data type for the parameter.
RECEIVE	’Y’ if the external function returns a value; ’N’ otherwise.

SYSADM.SYSFKCONSTRAINTS

This table contains each table’s foreign key columns.

Column Name	Description
CREATOR	The authorization ID of the user who created the table.
NAME	The name of the table to which foreign key applies.
CONSTRAINT	The name of the foreign key constraint.
FKCOLSEQNUM	The sequence number of the foreign key column within the foreign key.

Column Name	Description
REFSCOLUMN	The name of the column in the dependent table which has the foreign key.
REFDTBCREATOR	The authorization ID of the user who created the parent table referenced by the foreign key.
REFDTBNAME	The name of the parent table referenced by the foreign key.
REFDCOLUMN	The column name of the referenced column in the parent table.

SYSADM.SYSINDEXES

This table contains one row for every index, including indexes on system catalog tables.

Column Name	Description
NAME	The name of the index.
CREATOR	The authorization ID of the creator of the index.
TBNAME	The name of the table on which the index is defined.
TBCREATOR	The authorization ID of the creator of the table on which the index is defined.
UNIQUERULE	Indicates whether the index is unique: ‘D’ if duplicates are allowed and ‘U’ if the index is unique.
COLCOUNT	The number of columns in the index.
IXTYPE	Indicates the type of index: 'H' if hashed and 'B' if B+tree.
CLUSTERRULE	Indicates whether the index is clustered: 'Y' if clustered and 'N' if not.
SYSTEM	'Y' if the index is system-defined; ‘N’ if it is not.
IXSIZE	The number of rows in the table, as specified by the user.
PERCENTFREE	The amount of free space to leave on each index page when the index is first built. If not specified by the user, this defaults to 10 percent.

Column Name	Description
HEIGHT	The height of the index. Minimum value is 1, which indicates that the index has only one page. The height of the index tree, also called depth, is the number of nodes that have to be read from the root to the leaf level, inclusive. This statistic is maintained dynamically in the index's control page, but is only recorded in the SYSINDEXES table either when you first create the index, or run UPDATE STATISTICS.
INDEXPAGECOUNT	For a B+tree index, this is the number of index pages. For a hash index, this is null.
LEAFCOUNT	For a B+tree index, this is the number of nodes in the bottom leaves of the index (leaf page). It is also the number of pages in the index's sequence set. For a hash index, this is null.
CLUSTERCOUNT	For a B+tree index, this is the cluster count, which is the total number of page changes that would occur if the entire table was read through the index's sequence set. The minimum value (for a perfectly clustered B+tree index) is the number of data pages. The maximum value (for a completely unclustered index) is the number of rows in the table. For a hash index, this is null.
PRIMPAGECOUNT	For a B+tree index, this is null. For a hash index, this is the number of primary pages allocated for the index's subject table. This is the same as the number of hash slots available for distributing the table's row.
OVFLPAGECOUNT	For a B+ tree index, this is null. For a hash index, this is the number of overflow pages allocated for the index's subject table. When you first create the table and index, this is the number of overflow pages that SQLBase pre-allocates. Subsequently, this number increases as additional overflow pages are required to handle hashing collisions.
AVGKEYLEN	For a B+ tree index, this is the average key length across all index entries. This statistic is necessary because SQLBase maintains all index entries as variable length, minimum prefix only fields. For a hash index, this is null.

Column Name	Description
GROUPNUM	For a B+ tree index, this is the group number for index pages. For a hash index, this is the group number of overflow pages.
USECOUNT	The number of times the index has been used in locating data. When a table or index is used for the first time the current setting for this variable is loaded into memory. This memory variable is incremented every time a full index scan is used. This column is updated when the last cursor has disconnected from the database. To update the column, SQLBase establishes it own internal cursor to perform the update and committing of data. Since there are no users connected to the database all existing locks on the external dictionary will have been cleared and the update performs with no lock contentions.

SYSADM.SYSKEYS

This table contains one row for each column in an index.

Column Name	Description
IXNAME	The name of the index.
IXCREATOR	The authorization ID of the creator of the index.
COLNAME	The name of the column of the key.
COLNO	The numerical position of the column in the row; for example 2 (out of 7 columns in the table).
COLSEQ	The numerical position of the column in the key; for example 2 (out of 3 columns constituting the key).
ORDERING	The sort order of the column in the key. 'A' if ascending and 'D' if descending.
FUNCTION	The @function definition used to define the key, if you defined one; for example, CREATE INDEX EMPIX ON EMP (@UPPER (NAME)) displays the @UPPER(NAME) in this column.

Column Name	Description
DISTINCTCOUNT	The number of distinct keys for the portion of the key from the first column up to the COLNO value. SYSKEYS has one entry for each prefix key of a multi-column index key. The first entry contains the DISTINCTCOUNT value for the first column in the key. The second entry contains DISTINCTCOUNT values for first two columns in the key, and so on.

SYSADM.SYSOBAUTH

This table contains one row for each user granted execute privilege on an external function.

Column Name	Description
CREATOR	The authorization ID of the user who created the external function.
NAME	The name of the external function.
GRANTEE	The name of the user who has execute privilege on the external function.
OBJAUTH	Contains ‘E’ for execute to indicate the type of authorization for the object.
OBJTYPE	Contains ‘F’ for external function to indicate the object type.

SYSADM.SYSOBJSYN

This table contains one row for each synonym created for an external function, stored command, or stored procedure.

Column Name	Description
NAME	The synonym name.
CREATOR	The authorization ID of the user who created the synonym.
OBJNAME	The name of the object.
OBJCREATOR	The authorization ID of the user who created the object.

Column Name	Description
OBJTYPE	Contains ‘F’ for external function to indicate the object type. Contains ‘P’ for stored procedures and stored commands.

SYSADM.SYSPARTTRANS

This table contains information about in-doubt distributed transactions. Once a transaction is resolved, the row is removed from the table.

This table does not contain information about all in-doubt transactions while the database is active; it only contains rows about transactions that are in-doubt after a recovery.

Column Name	Description
ID1	The global transaction ID.
ID2	Reserved for future use.
STATE	The current status of the transaction. It can be any of the following: • IDLE • PREPARED • COMMITTING • ABORTING
PROTOCOL	The protocol being used for the two-phase commit. Currently, SQLBase only uses the STANDARD protocol.
LASTMODTIME	The timestamp indicating when the transaction status was last modified.
DB	The name of the commit server.
USERNAME	The user name that is connected to the commit server.
PASSWORD	The (encrypted) password for connecting to the commit server.

Column Name	Description
SOURCE	A string indicating how transaction information was entered into this table. Valid values are: • NORMAL Indicates row was inserted while the database was active. • CRASH Indicates row was inserted just after crash recovery. • ROLLFORWARD Indicates row was inserted just after rollforward recovery.

SYSADM.SYSPKCONSTRAINTS

This table contains the primary key columns of a table.

Column Name	Description
CREATOR	The authorization ID of the user who created the table.
NAME	The name of the table to which the primary key applies.
PKCOLSEQNUM	The relative column number of the primary key column.
COLNAME	The name of the column to which the primary key applies.

SYSADM.SYSROWIDLISTS

This table lists information about result sets, and contains one row for each row ID. If SET RESTRICTION is set to ON, this table lists information only about the user's result sets.

Column Name	Description
NAME	Name of row id list (result set) saved.
CREATOR	Name of creator of above result set.

SYSADM.SYSSYNONYMS

This table contains one row for each synonym of a table or view.

Column Name	Description
NAME	Synonym for the table or view.
CREATOR	The authorization ID of the creator of the synonym.
TBNAME	The name of the table or view.
TBCREATOR	The authorization ID of the creator of the table or view.

SYSADM.SYSTABAUTH

This table contains the privileges of users for tables or views.

Column Name	Description
GRANTEE	The authorization ID of the user who holds the privileges described in this row.
TCREATOR	The authorization ID of the user who created the table or view on which privileges are held.
TTNAME	The name of the table or view on which privileges are held. 'T' stands for target.
UPDATECOLS	Contains a '*' if the user has update privileges on only some of the columns in the target table. The column names for which privileges are held are contained in the SYSCOLAUTH table. If the user holds no update privileges or if update is allowed on the entire table, then this column contains a blank.
ALTERAUTH	Contains a 'Y' if the user is allowed to alter the target table. Otherwise it contains a blank.
DELETEAUTH	Contains a 'Y' if the user is allowed to delete rows from the target table or view. Otherwise it contains a blank.
INDEXAUTH	Contains a 'Y' if the user is allowed to create or drop indexes on the target table. Otherwise it contains a blank.
INSERTAUTH	Contains a 'Y' if the user is allowed to insert rows into the target table or view. Otherwise it contains a blank.
SELECTAUTH	Contains a 'Y' if the user is allowed to read rows from the target table or view. Otherwise it contains a blank.

Column Name	Description
UPDATEAUTH	Contains a 'Y' if the user is allowed to update rows in the target table or view. Otherwise it contains a blank.

SYSADM.SYSTABCONSTRAINTS

This table contains the table constraints.

Column Name	Description
CREATOR	The authorization ID of the user who created the table.
NAME	The name of the table to which the constraint applies.
CONSTRAINT	The name of the constraint which is being used by the database. This column also shows the name of the foreign key. For the primary key, it is set to 'PRIMARY'.
TYPE	The type of constraint. This column is set to 'P' for a primary key, and 'F' for a foreign key constraint.
DELETERULE	The DELETE rule being followed by the database. This column is not used for primary key constraints. This column can be set to: CASCADE C SET NULL N RESTRICT R The default is RESTRICT.
USRERRINSDEP	The user-specified error message number for an INSERT_DEPENDENT violation. If there was no user-specified error, this column is set to 0.
USRERRUPDDEP	The user-specified error message number for an UPDATE_DEPENDENT violation. If there was no user-specified error, this column is set to 0.
USRERRDELPAR	The user-specified error message number for a DELETE_PARENT violation. If there was no user-specified error, this column is set to 0.
USRERRUPDPAR	The user-specified error message number for an UPDATE_PARENT violation. If there was no user-specified error, this column is set to 0.

SYSADM.SYSTABLES

This table contains one row for each table or view.

Column Name	Description
CREATOR	The authorization ID of the user who created the table or view.
NAME	The name of the table or view.
COLCOUNT	The number of columns in the table or view.
REMARKS	A user-specified comment about each row. The maximum length of a comment is 254 characters.
TYPE	'T' if the row describes a table or 'V' if the row describes a view.
SYSTEM	'Y' if the table is system-defined; ‘N’ if it is not.
SNUM	A unique serial number that serves as a table ID. This is used by the UNLOAD DATABASE command.
LABEL	A user-specified label about each table and row. The maximum length of a label is 30 characters.
PERCENTFREE	The amount of free space left in each table row page when it is initially filled. The default value is 10 percent.
ROWCOUNT	The number of rows in the table. This is maintained dynamically in the table’s control page, but is only recorded in this table when you execute UPDATE STATISTICS.
PAGECOUNT	The number of base row pages in the table. This column is updated each time you run UPDATE STATISTICS. If you do not set any values yourself with the SET clause of this command, this column is the sum of the ROWPAGECOUNT, EXTENTPAGECOUNT, and LONGPAGECOUNT values. Initial default value is 2.

Column Name	Description
ROWPAGECOUNT	The number of base row pages in the table. This includes any extent pages that may also be allocated. This is recorded and updated only when you run UPDATE STATISTICS. Initial default value is 2.
LONGPAGECOUNT	The number of pages in the table allocated to store LONG VARCHAR columns. This is recorded and updated only when you run UPDATE STATISTICS.
EXTENTPAGECOUNT	The number of extent pages.
FREESLOTS	The total number of free slots in all data pages. When a new page is allocated to a table, there is one entry in the slot table, which is called a free slot. As new rows are inserted into the page, more entries are added to the slot-tables and the space for each additional slot is carved out of the free slot. When a row is deleted its slot becomes free.
USEDSPACE	The total number of used bytes in all data pages, including user data, page header, etc.
FREESPACE	The total number of free bytes in all data pages used by free slots (holes). A large ratio of freespace to the usedspace calls for either a database reorganization or a load/unload. One possible reason for this can be that the average base row length is too large. The ideal ratio of freespace to the used space is small.
AVGROWLEN	The average row length in the table. This can differ significantly from the defined row length because SQLBase stores all columns as variable data regardless of the data type used in the column definition. Note that this is the row length that is stored on the base table pages, and therefore excludes all LONG VARCHAR columns.
AVGROWLONGLEN	The average length of all LONGVARCHAR columns stored in the table. This is zero (0) if no LONGVARCHAR columns exist.
GROUPNUM	The group number for the table.

Column Name	Description
TABLESCAN	The number of times a full table scan was performed on the table. When a table or index is used for the first time the current setting for this variable is loaded into memory. This memory variable is incremented every time a full table scan is used. This column is updated when the last cursor has disconnected from the database. To update the column, SQLBase establishes its own internal cursor to perform the update and committing of data. Since there are no users connected to the database all existing locks on the external dictionary will have been cleared and the update performs with no lock contentions.

SYSADM.SYSTRGCOLS

This table contains one row for every column on which an update trigger exists.

Column Name	Description
CREATOR	The authorization ID of the creator of the trigger.
NAME	The name of the trigger.
TBSNUM	The serial number of the trigger table.
ACTIONTIME	Whether the trigger is defined to activate before or after the attempted modification.
COLNO	The column’s relative position in the trigger table (starting with 1).

SYSADM.SYSTRIGGERS

This table contains one row for every trigger.

Column Name	Description
CREATOR	The authorization ID of the creator of the trigger.
NAME	The name of the trigger.
TBSNUM	The serial number of the trigger table.
ACTIONTIME	Whether the trigger is defined to activate before or after the attempted modification.

Column Name	Description
TRIGGEREVENT	The event (INSERT, UPDATE, or DELETE) on which the trigger is designed to activate.
OLDVALUENAME	The alias for the table containing the old column values.
NEWVALUENAME	The alias for the table containing the new column values.
FREQUENCY	Whether the trigger is defined to activate once for each row or for each statement.
SYSTEM	'Y' if the trigger is system-defined; ‘N’ if it is not.
SPSNUM	An integer column that refers to the serial number (SNUM in SYSCOMMANDS) of the stored procedure that is executed by a trigger.
TEXT	The text of the stored or inline procedure.
NUMCOLUMNS	The number of columns named in the UPDATE OF clause. The default is zero (0) which means that the trigger is activated by a modification attempt on any column. This is zero (0) for triggers defined on INSERT and DELETE operations.
ORDERVALUE	The order sequence number of a trigger. The sequence number determines the order in which triggers are fired within a table, by event (using the INSERT, UPDATE, and DELETE commands), time (BEFORE and AFTER), and frequency (using STATEMENT and ROW).
STATUS	'Y' if the trigger is enabled; 'N' if it is enabled.

SYSADM.SYSUSERAUTH

This table contains the database authority level of each user.

Column Name	Description
NAME	The authorization ID of each valid user of the database. A user is valid if he has CONNECT authority.
RESOURCEAUTH	'Y' if the user can create or drop tables or 'G' if the user is SYSADM. Otherwise blank.

Column Name	Description
DBAAUTH	'Y' if the user has DBA authority, a 'G' if the user is SYSADM. Otherwise blank.
PASSWORD	The (encrypted) password associated with the authorization ID.

SYSADM.SYSVIEWS

This table contains one or more rows for each view.

Column Name	Description
NAME	The name of the view.
CREATOR	The authorization ID of the creator of the view.
SEQNO	The sequence number of this row. The first text portion of the view is on row one and successive rows have increasing values of SEQNO.
CHECKFLAG	Whether the CHECK option was specified in the CREATE VIEW command; 'Y' means yes and 'N' means no.
TEXT	The text of the CREATE VIEW command. This column is 250 characters long. If the CREATE VIEW command is longer than 40 characters, then each row contains a portion of the text.

System table operations

The system catalog tables are all owned by SYSADM. Users with the proper privileges can perform the following operations on system tables:

- Select rows
- Create a view
- Create a synonym
- Create an index on a column
- Add user-defined columns
- Drop user-defined columns
- Update user-defined columns

You cannot DROP a system table or system-defined column, nor can you INSERT or DELETE rows into or from a system table.

The privilege to perform any of the above operations must be granted explicitly to a user by the SYSADM or by a user with DBA authority. Users who do not have these privileges cannot access the system tables directly.

System catalog views

You can create views of the system tables for users. Use the keyword USER in the view definition's search condition. For example, a view called TABLES could contain the names of all the tables that can be accessed with a given authorization ID; a view called COLUMNS could contain the names of all the columns in those tables, and so forth. SYSADM would own these views and other users could select from them but could not modify them.

Although you can add columns to a system catalog table, SQLBase does not maintain them. For example, the UNLOAD command ignores user-defined columns; it does not unload them.

Server-independent system catalog views

Your applications can use the following views on the system catalog tables to access catalog information in a server-independent fashion.

These views are a common subset of the system catalog tables supported by most SQL database vendors. Commands such as LOAD and UNLOAD use these views to access non-SQLBase databases (such as DB2).

View Name	Columns
SYSSQL.SYSCOLAUTH	GRANTEE CREATOR TNAME COLNAME
SYSSQL.SYSCOLUMNS	TBCREATOR NAME TBNAME COLNO COLTYPE LENGTH NULLS UPDATES REMARKS SCALE LABEL

View Name	Columns
SYSSQL.SYSCOMMANDS	CREATOR NAME TYPE SYSTEM VALID AUTORECOMPILE TEXT
SYSSQL.SYSDEPENDENCIES	DETCREATOR DETNAME DETTYPE DEPCREATOR DEPNAME DEPTYPE
SYSSQL.SYSEVENTS	CREATOR NAME TYPE TYPEID EVENTID BEGINTIME INTERVAL TEXT
SYSSQL.SYSEXECUTEAUTH	CREATOR NAME GRANTEE USECREATORPRIV USEGRANTEEPRIV
SYSSQL.SYSINDEXES	TBCREATOR NAME TBNAME CREATOR UNIQUERULE COLCOUNT IXTYPE CLUSTERRULE IXSIZE PERCENTFREE
SYSSQL.SYSKEYS	IXCREATOR IXNAME COLNAME COLNO COLSEQ ORDERING FUNCTION

View Name	Columns
SYSSQL.SYSOBJAUTH	CREATOR NAME GRANTEE OBJAUTH OBJTYPE
SYSSQL.SYSOBJSYN	NAME CREATOR OBJNAME OBJCREATOR OBJTYPE
SYSSQL.SYSPARTTRANS	ID STATE PROTOCOL LASTMODTIME DB USERNAME PASSWORD
SYSSQL.SYSSYNONYMS	NAME CREATOR TBNAME TBCREATOR
SYSSQL.SYSTABAUTH	GRANTEE TCREATOR TTNAME UPDATECOLS ALTERAUTH DELETEAUTH INDEXAUTH INSERTAUTH SELECTAUTH UPDATEAUTH
SYSSQL.SYSTABLES	CREATOR NAME COLCOUNT TYPE REMARKS PERCENTFREE LABEL

MAIN database system tables

In addition to the standard system catalog tables, the MAIN database contains system catalog tables specific to partitioned databases and distributed transactions. These tables are owned by SYSADM and maintained by SQLBase.

Table Name	Description
CSVPARTS	Contains information about all the participants in a distributed transaction.
CSVTRANS	Used by the distributed transaction's commit server to handle two-phase commit.
DEFAULTS	Lists default storage group.
DATABASES	Lists all partitioned databases.
STOGROUPS	Lists storage groups.
AREAS	Lists database areas.
STOAREAS	Lists database areas and their storage groups.
EXTENTS	Lists extents that have been used.
FREEEXTS	Lists extents that are available.
CSVTRANSV	Internal use only.
CSVPARTSV	Internal use only.

SYSADM.CSVPARTS

This table contains information about all the participants in a transaction.

Column Name	Description
ID1	The global transaction ID.
ID2	Reserved for future use.
DB	The name of the commit server.
USERNAME	The user name that is connected to the commit server.
PASSWORD	The (encrypted) password for connecting to the commit server.

SYSADM.CSVTRANS

This table is used by the commit server to record general information about the two-phase commit operation. There is one row for every distributed transaction.

Column Name	Description
ID1	The global transaction ID.
ID2	Reserved for future use.
STATE	The current status of the transaction. It can be any of the following: • IDLE • PREPARED • COMMITTING • ABORTING
PROTOCOL	The protocol being used for the two-phase commit. Currently, SQLBase only uses the STANDARD protocol.
NUMPARTS	The number of participants.
COMMITTIME	Indicates when the transaction was committed.
LASTMODTIME	A timestamp indicating when the transaction status was last modified.

SYSADM.DEFAULTS

This table contains information about default storage groups.

Column Name	Description
STOGROUP	Default storage group name.

SYSADM.DATABASES

This table contains information about partitioned databases.

Column Name	Description
NAME	Partitioned database name.
STOGROUP	Storage group associated with the database.
LOGSTOGROUP	Storage group associated with the log files.

SYSADM.AREAS

This table contains information about database areas.

Column Name	Description
NAME	Area name.
AREAID	Internal use only.
PATHNAME	Path of the area file or device.
AREASIZE	Size of the area in bytes. Note that for the view, the size is in megabytes.

SYSADM.STOGROUPS

This table contains information about storage groups.

Column Name	Description
NAME	Storage group name.

SYSADM.STOAREAS

This table contains information about storage groups in association with areas.

Column Name	Description
STOGROUP	Storage group name.
AREANAME	Database area name.

SYSADM.EXTENTS

This table contains information about database extents that have been used.

Column Name	Description
AREAID	Internal use only.
OFFSET	Offset within the area, in bytes. Note that for the view, the size is in megabytes.
EXTSIZE	Size of the extent, in bytes. Note that for the view, the size is in megabytes.
DBNAME	Name of the database associated with the extent.
FILENAME	Internal use only.

Column Name	Description
FILEPOS	Internal use only.

SYSADM.FREEEXTS

This table contains information about available database extents.

Column Name	Description
AREAID	Internal use only.
OFFSET	Offset within the area, in bytes. Note that for the view, the size is in megabytes.
EXTSIZE	Size of the free extent, in bytes. Note that for the view, the size is in megabytes.

MAIN database system views

The MAIN database contains system catalog views specific to distributed transactions and partitioned databases. These views are also owned by SYSSQL and maintained by SQLBase.

Table Name	Description
CSVPARTS	ID DB USERNAME
CSVTRANS	ID STATE PROTOCOL NUMPARTS COMMITTIME LASTMODTIME
DEFAULTS	STOGROUP
DATABASES	NAME STOGROUP LOGSTOGROUP
STOGROUPS	NAME
AREAS	NAME AREASIZE PATHNAME

Table Name	Description
STOAREAS	STOGROUP AREANAME
EXTENTS	NAME DBNAME EXTSIZE OFFSET
FREEEXTS	OFFSET EXTSIZE

MAIN stored commands

These stored commands manipulate the information in the tables of the MAIN database.

DATABASES table:

```
store selname select name from databases
      where name = :dbname;
store seldb select stogroup from databases
      where name = :dbname;
store sellog select logstogroup from databases
      where name = :dbname;
```

EXTENTS table:

```
store selsize select sum(extsize) from extents
      where dbname = :dbname and filename =
      :filename;
store selexts select areaid, offset, extsize
      from extents
      where dbname = :dbname and filename =
      :filename
      order by filepos;
store selexta select areaid, offset, extsize
      from extents
      where dbname = :dbname and filename =
      :filename
      order by areaid, offset;
store delfile delete from extents where dbname =
      :dbname
      and filename = :filename;
```

```
store insext insert into extents values
      (:areaid, :offset, :extsize,
      :dbname, :filename, :filepos);
```

AREAS table:

```
store selarea select pathname, areasize from
      areas where areaid = :areaid;
```

FREEEXTS table:

```
store selgt select f.areaid, f.offset, f.extsize
      from freeexts f, areas a, stoareas s
      where f.areaid > :areaid and f.extsize >=
      :extsize
      and s.stogroup = :stogroup and s.areaname
      = a.name
      and a.areaid = f.areaid order by f.areaid,
      f.extsize;
store selleq select f.areaid, f.offset,
      f.extsize
      from freeexts f, areas a, stoareas s
      where f.areaid <= :areaid and f.extsize >=
      :extsize
      and s.stogroup = :stogroup and s.areaname
      = a.name
      and a.areaid = f.areaid order by f.areaid,
      f.extsize;
store selsto select f.areaid, f.offset,
      f.extsize
      from freeexts f, areas a, stoareas s
      where f.extsize >= :extsize and s.stogroup
      = :stogroup
      and s.areaname = a.name and a.areaid =
      f.areaid
      order by f.extsize;
store selsame select offset, extsize from
      freeexts
      where areaid = :areaid and extsize >=
      :extsize
      order by extsize;
store delfree delete from freeexts where areaid
      = :areaid and
      offset = :offset;
```

```
store updfree update freeexts set offset =
      :newoffset,
      extsize = :newextsize where areaid =
      :areaid and
      offset = :offset;
store selcomb select offset, extsize from
      freeexts
      where areaid = :areaid and (offset +
      extsize = :offset or
      offset = :endoffset) order by offset;
store insfree insert into freeexts values
      (:areaid, :offset, :extsize);
```

Client stored commands

```
store selaid select areaid from areas where areaid
      = :1;
store insare insert into areas values (:1, :2, :3,
      :4);
store insfex insert into freeexts values(:1, :2,
      :3);
store selasf select areasize, pathname from areas
      where name = :1;
store seleps select areas.areaid from areas,
      extents
      where extents.offset + extents.extsize > :1
      and areas.name = :2 and areas.areaid =
      extents.areaid;
store selasz select areasize from areas where name
      = :1;
store updasz update areas set areasize = :1 where
      name = :2;
store selsta select areaname from stoareas where
      areaname = :1;
store selexa select extents.areaid from extents,
      areas
      where areas.areaid = extents.areaid and
      areas.name = :1;
store delfex delete from freeexts
      where areaid = (select areaid from areas
      where name = :1);
```

```
store delare delete from areas where name = :1;
store insstg insert into stogroups values (:1);
store inssta insert into stoareas values (:1, :2);
store delsta delete from stoareas where stogroup =
      :1 and
      areaname = :2;
store selsgd select stogroup from databases where
      stogroup = :1;
store seldfs select stogroup from defaults;
store seldfl select stogroup from defaults where
      stogroup = :1;
store delstg delete from stogroups where name = :1;
store delasa delete from stoareas where stogroup =
      :1;
store deldfl delete from defaults;
store insdfl  insert into defaults values (:1);
store selsgp select name from stogroups where name
      = :1;
store insdbs insert into databases values (:1, :2,
      :3);
store upddbsd update databases set stogroup = :1
      where name = :2;
store upddbsl update databases set logstogroup = :1
      where name =:2;
store deldbs delete from databases where name = :1;
store seldbs select name from databases where name
      = :1;
store selexf select filename from extents where
      dbname = :1;
store seladb select name from databases;
store selare select areaid from areas where name =
      :1;
store updfxs update freeexts set extsize = extsize
      - :1
      where areaid = :2 and offset + extsize > :3;
store delfxs delete from freeexts where extsize =
      0;
```

Appendix B

SQLBase Procedures

This appendix lists the SQLBase-supplied procedures. SQLBase supplies the RECOMPILE and SYSADM.SYSPROC_ALTTABTRIG procedures.

SQLBase-supplied procedures

The SQLBase-provided procedures are not part of the *start.dbs* template database. Instead, the procedures are installed under a specific directory. Read the steps below for details.

Loading SQLBase-supplied procedures

Note: To load SQLBase-supplied procedures, you must have SYSADM authority.

1. Start your server.
2. Start SQLTalk and connect to the database where you want to load the stored procedure(s).
3. Go to the following directory to locate the stored procedure(s) you want to load:
    - *recomp.sql* is found in the *sysproc* directory (This directory is installed under the directory where you install the client software.)
    - *rep_trig.sql* is found in the *Centura* directory
4. Run the *recomp.sql* and *rep_trig.sql* scripts in the SQLTalk window or at the SQL> command line prompt. These scripts “load” the procedures to your specified database. The procedures are installed during SQLBase installation.

You must run *recomp.sql* or *rep_trig.sql* against each database that requires the use of the stored procedure. For example, run *rep_trig.sql* against all databases that contain the triggers you want to enable or disable. Note that you only need to run the script once for each database. You must be in the Centura home directory (the default) when you run any procedure load scripts.

RECOMPILE procedure

Use the RECOMPILE stored procedure to manually recompile a stored command. For information on stored procedures, see the *Procedures, Triggers, and Events* chapter in the *SQL Language Reference*.

You should run the RECOMPILE procedure when a stored command is invalidated. A stored command becomes invalid when you alter its associated tables, such as when you drop an index or column. Even if the table change does not directly affect the stored command, SQLBase still invalidates it. You cannot use the stored command until you replace or recompile it.

The RECOMPILE procedure requires the following parameters:

- User id

  This is the user name associated with the stored command.

- Stored command name

  Enter the command name in capital letters. You can use the underscore (_) and percent (%) wildcard characters for pattern matching.

- Valid/Invalid flag

  Enter **N** to designate invalid commands, or **Y** to designate valid commands. For more information on this flag, read the information under the SYSADM.SYSCOMMANDS table in *Appendix A, System Catalog Tables*.

- Input/Output values

  These are a stored procedure's receive parameters as described in the *Procedures, Triggers, and Events* chapter in the *SQL Language Reference*. There are three parameters, and each requires a value. You can enter a dummy value (for example, 0) to satisfy the execution requirement.

Unlike ALTER COMMAND, the EXECUTE RECOMPILE command actually recompiles a stored command. ALTER COMMAND sets the AUTORECOMPILE flag to ON; the stored command is not recompiled until it is executed and then only if it is invalid.

The RECOMPILE procedure does not alter the stored command's AUTORECOMPILE attribute (ON/OFF). You do not need to recompile system commands; SQLBase automatically recompiles them itself.

SYSADM.SYSPROC_ALTTABTRIG procedure

Use the SYSADM.SYSPROC_ALTTABTRIG procedure to enable or disable all triggers defined on a table. This procedure requires the following parameters:

- User id

  This is the user name associated with the table.

- Table name

  Enter the table name in capital letters. You can use the underscore (_) and percent (%) wildcard characters for pattern matching.

- Trigger status

  To change the status of the existing triggers defined on the table, Enter ether ENABLE or DISABLE.

- Input/Output values

  These are a stored procedure's receive parameters as described in the *Procedures, Triggers, and Events* chapter in the *SQL Language Reference*. The procedure returns the number of triggers enabled/disabled by this execution of the procedure. You can enter a dummy value (for example, 0) to satisfy the execution requirement.

Appendix C

System Utility Tables

This Appendix contains descriptions of the SQLBase system utility tables.

System utilities table summary

The system utility tables are used for certain internal operations of SQLBase. A system utility table does not contain any database object definitions and the rows in the table are not created by any explicit DDL statements. The tables are owned by SYSADM and maintained by SQLBase.

Table Name	Description
SYSCOMMITORDER	Used for logging transaction commit logs

SYSCOMMITORDER

This table is used for logging transaction commit logs. If you call the *sqlset* function and specify the SQLPCLG parameter, you can turn commit logging on. For details, read the description of the *sqlset* function in the *SQL Application Programming Interface Reference*.

When commit logging is on, every database transaction in which data was modified is logged in the SYSCOMMIT ORDER table.

Note: This table is primarily for internal use and should not be modified. You cannot drop this table or delete columns from it. If you are using SQLBase for Centura replication, manipulating this table or the SQLPCLG parameter may result in erroneous replication behavior.

Column Name	Description
CommitSequence	The sequence number for the committing transaction.
TransactionID	The transaction ID of the committing transaction.

Glossary

access path—The path used to get the data specified in a SQL command. An access path can involve an index or a sequential search (table scan), or a combination of the two. Alternate paths are judged based on the efficiency of locating the data.

aggregate function—A SQL operation that produces a summary value from a set of values.

alias—An alternative name used to identify a database object.

API (application programming interface)—A set of functions that a program uses to access a database.

application—A program written by or for a user that applies to the user's work. A program or set of programs that perform a task. For example, a payroll system.

argument—A value entered in a command that defines the data to operate on or that controls execution. Also called parameter or operand.

arithmetic expression—An expression that contains operations and arguments that can be reduced to a single numeric value.

arithmetic operator—A symbol used to represent an arithmetic operation, such as the plus sign (+) or the minus sign (-).

attribute—A characteristic or property. For example, the data type or length of a row. Sometimes, attribute is used as a synonym for column or field.

audit file—A log file that records output from an audit operation.

audit message—A message string that you can include in an audit file

audit operation—A SQLBase operation that logs database activities and performance, writing output to an audit file. For example, you can monitor who logs on to a database and what tables they access, or record command execution time.

authorization—The right granted to a user to access a database.

authorization-ID—A unique name that identifies a user. Associated to each authorization-id is a password. Abbreviated auth-id. Also called username.

back-end—See database server.

backup—To copy information onto a diskette, fixed disk, or tape for record keeping or recovery purposes.

base table—The permanent table on which a view is based. A base table is created with the CREATE TABLE command and does not depend on any other table. A base table has its description and its data physically stored in the database. Also called underlying table.

bindery—A NetWare 3.x database that contains information about network resources such as a SQLBase database server.

bind variable—A variable used to associate data to a SQL command. Bind variables can be used in the VALUES clause of an INSERT command, in a WHERE clause, or in the SET clause of an UPDATE command. Bind variables are the mechanism to transmit data between an application work area and SQLBase. Also called into variable or substitution variable.

browse—A mode where a user queries some of a database without necessarily making additions or changes. In a browsing application, a user needs to examine data before deciding what to do with it. A browsing application allows the user to scroll forward and backward through data.

buffer—A memory area used to hold data during input/output operations.

C/API—A language interface that lets a programmer develop a database application in the C programming language. The C/API has functions that a programmer calls to access a database using SQL commands.

cache—A temporary storage area in computer memory for database pages being accessed and changed by database users. A cache is used because it is faster to read and write to computer memory than to a disk file.

Cartesian product—In a join, all the possible combinations of the rows from each of the tables. The number of rows in the Cartesian product is equal to the number of rows in the first table times the number of rows in the second table, and so on. A Cartesian product is the first step in joining tables. Once the Cartesian product has been formed, the rows that do not satisfy the join conditions are eliminated.

cascade—A delete rule which specifies that changing a value in the parent table automatically affects any related rows in the dependent table.

case sensitive—A condition in which names must be entered in a specific lower-case, upper-case, or mixed-case format to be valid.

cast—The conversion between different data types that represent the same data.

CHAR—A column data type that stores character strings with a user-specified length. SQLBase stores CHAR columns as variable-length strings. Also called VARCHAR.

character—A letter, digit, or special character (such as a punctuation mark) that is used to represent data.

character string—A sequence of characters treated as a unit.

checkpoint—A point at which database changes older than the last checkpoint are flushed to disk. Checkpoints are needed to ensure crash recovery.

clause—A distinct part of a SQL command, such as the WHERE clause; usually followed by an argument.

client—A computer that accesses shared resources on other computers running as servers on the network. Also called front-end or requester.

column—A data value that describes one characteristic of an entity. The smallest unit of data that can be referred to in a row. A column contains one unit of data in a row of a table. A column has a name and a data type. Sometimes called field or attribute.

command—A user request to perform a task or operation. In SQLTalk, each command starts with a name, and has clauses and arguments that tailor the action that is performed. A command can include limits or specific terms for its execution, such as a query for names and addresses in a single zip code. Sometimes called statement.

commit—A process that causes data changed by an application to become part of the physical database. Locks are freed after a commit (except when cursor-context preservation is on). Before changes are stored, both the old and new data exist so that changes can be stored or the data can be restored to its prior state.

commit server—A database server participating in a distributed transaction, that has commit service enabled. It logs information about the distributed transaction and assists in recover after a network failure.

composite primary key—A primary key made up of more than one column in a table.

concatenated key—An index that is created on more than one column of a table. Can be used to guarantee that those columns are unique for every row in the table and to speed access to rows via those columns.

concatenation—Combining two or more character strings into a single string.

concurrency—The shared use of a database by multiple users or application programs at the same time. Multiple users can execute database transactions simultaneously without interfering with each other. The database software ensures that all users see correct data and that all changes are made in the proper order.

configure—To define the features and settings for a database server or its client applications.

connect—To provide a valid authorization-id and password to log on to a database.

connection handle—Used to create multiple, independent connections. An application must request a connection handle before it opens a cursor. Each connection handle represents a single transaction and can have multiple cursors. An application may request multiple connection handles if it is involved in a sequence of transactions.

consistency—A state that guarantees that all data encountered by a transaction does not change for the duration of a command. Consistency ensures that uncommitted updates are not seen by other users.

constant—Specifies an unchanging value. Also called literal.

control file—An ASCII file containing information to manage segmented load/unload files.

cooperative processing—Processing that is distributed between a client and a server in a such a way that each computer works on the parts of the application that it is best at handling.

coordinator—The application that initiates a distributed transaction.

correlated subquery—A subquery that is executed once for each row selected by the outer query. A subquery cannot be evaluated independently because it depends on the outer query for its results. Also called a repeating query. Also see subquery and outer query.

correlation name—A temporary name assigned to a table in an UPDATE, DELETE, or SELECT command. The correlation name and column name are combined to refer to a column from a specific table later in the same command. A correlation name is used when a reference to a column name could be ambiguous. Also called range variable.

crash recovery—The procedures that SQLBase uses automatically to bring a database to a consistent state after a failure.

current row—The latest row of the active result set which has been fetched by a cursor. Each subsequent fetch retrieves the next row of the active result set.

cursor—The term cursor refers to one of the following definitions:

- The position of a row within a result table. A cursor is used to retrieve rows from the result table. A named cursor can be used in the CURRENT OF clause or the ADJUSTING clause to make updates or deletions.
- A work space in memory that is used for gaining access to the database and processing a SQL command. This work space contains the return code, number of rows, error position, number of select list items, number of bind variables, rollback flag, and the command type of the current command.
- When the cursor belongs to an explicit connection handle that is created using the SQL/API function call *sqlcch* or the SQLTalk BEGIN CONNECTION command, it identifies a task or activity within a transaction. The task or activity can be compiled/executed independently within a single connection thread.

  Cursors can be associated with specific connection handles, allowing multiple transactions to the same database within a single application. When this is implemented, only one user is allowed per transaction.
- When a cursor belongs to an implicit connection handle created using the SQL/API function call *sqlcnc* or *sqlcnr*, or the SQLTalk CONNECT command, the cursor applies to an application in which you are connecting the cursor to a specific database that belongs to a single transaction.

cursor-context preservation—A feature of SQLBase where result sets are maintained after a COMMIT. A COMMIT does not destroy an active result set (cursor context). This enables an application to maintain its position after a COMMIT, INSERT, or UPDATE. For fetch operations, locks are kept on pages required to maintain the fetch position.

cursor handle—Identifies a task or activity within a transaction. When a connection handle is included in a function call to open a new cursor, the function call returns a cursor handle. The cursor handle can be used in subsequent SQL/API calls to identify the connection thread. A cursor handle is always part of a specific transaction and cannot be used in multiple transactions. However, a cursor handle can be associated with a specific connection handle. The ability to have multiple transactions to the same database within a single application is possible by associating cursor handles with connection handles.

Cursor Stability (CS)—The isolation level where a page acquires a shared lock on it only while it is being read (while the cursor is on it). A shared lock is dropped as the cursor leaves the page, but an exclusive lock (the type of lock used for an update) is retained until the transaction completes. This isolation level provides higher concurrency than Read Repeatability, but consistency is lower.

data dictionary—See system catalog.

data type—Any of the standard forms of data that SQLBase can store and manipulate. An attribute that specifies the representation for a column in a table. Examples of data types in SQLBase are CHAR (or VARCHAR), LONG VARCHAR (or LONG), NUMBER, DECIMAL (or DEC), INTEGER (or INT), SMALLINT, DOUBLE PRECISION, FLOAT, REAL, DATETIME (or TIMESTAMP), DATE, TIME.

database—A collection of interrelated or independent pieces of information stored together without unnecessary redundancy. A database can be accessed and operated upon by client applications such as SQLTalk.

database administrator (DBA)—A person responsible for the design, planning, installation, configuration, control, management, maintenance, and operation of a DBMS and its supporting network. A DBA ensures successful use of the DBMS by users.

A DBA is authorized to grant and revoke other users' access to a database, modify database options that affect all users, and perform other administrative functions.

database area—A database area corresponds to a file. These areas can be spread across multiple disk volumes to take advantage of parallel disk input/output operations.

database management system (DBMS)—A software system that manages the creation, organization, and modification of a database and access to data stored within it. A DBMS provides centralized control, data independence, and complex physical structures for efficient access, integrity, recovery, concurrency, and security.

database object—A table, view, index, synonym or other object created and manipulated through SQL.

database server—A DBMS that a user interacts with through a client application on the same or a different computer. Also called back-end.

DATE—A column data type in SQL that represents a date value as a three-part value (day, month, and year).

date/time value—A value of the data type DATE, TIME, or TIMESTAMP.

DCL (Data Control Language)—SQL commands that assign database access privileges and security such as GRANT and REVOKE.

DDL (Data Definition Language)—SQL commands that create and define database objects such as CREATE TABLE, ALTER TABLE, and DROP TABLE.

deadlock—A situation when two transactions, each having a lock on a database page, attempt to acquire a lock on the other's database page. One type of deadlock is where each transaction holds a shared lock on a page and each wishes to acquire an exclusive lock. Also called deadly embrace.

DECIMAL—A column data type that contains numeric data with a decimal point. Also called DEC.

default—An attribute, value, or setting that is assumed when none is explicitly specified.

delimited identifier—An identifier enclosed between two double quote characters (“) because it contains reserved words, spaces, or special characters.

delimiter—A character that groups or separates items in a command.

dependent object—An object whose existence depends on another object.

For example, if a stored procedure calls an external function, the stored procedure is the dependent object of the external function, since its existence depends on the external function.

dependent table—The table containing the foreign key.

determinant object—An object that determines the existence of another object.

For example, if a stored procedure calls an external function, the external function is the determinant object, since it determines the existence of the stored procedure.

dirty page—A database page in cache that has been changed but has not been written back to disk.

distributed database—A database whose objects reside on more than one system in a network of systems and whose objects can be accessed from any system in the network.

distributed transaction—Coordinates SQL statements among multiple databases that are connected by a network.

DLL (Dynamic Link Library)—A program library written in C or assembler that contains related modules of compiled code. The functions in a DLL are not read until run-time (dynamic linking).

DML (Data Manipulation Language)—SQL commands that change data such as INSERT, DELETE, UPDATE, COMMIT, and ROLLBACK.

DOUBLE PRECISION—A column data type that stores a floating point number.

DQL (Data Query Language)—The SQL SELECT command, which lets a user request information from a database.

duplicates—An option used when creating an index for a table that specifies whether duplicate values are allowed for a key.

embedded SQL—SQL commands that are embedded within a program, and are prepared during precompilation and compilation before the program is executed. After a SQL command is prepared, the command itself does not change (although values of host variables specified within the command can change). Also called static SQL.

entity—A person, place, or thing represented by a table. In a table, each row represents an entity.

equijoin—A join where columns are compared on the basis of equality, and all the columns in the tables being joined are included in the results.

Ethernet—A LAN with a bus topology (a single cable not connected at the ends). When a computer wants to transmit, it first checks to see if another computer is transmitting. After a computer transmits, it can detect if a collision has happened. Ethernet is a broadcast network and all computers on the network hear all transmissions. A computer selects only those transmissions addressed to it.

exclusive lock (X-lock)—An exclusive lock allows only one user to have a lock on a page at a time. An exclusive lock prevents another user from acquiring a lock until the exclusive lock is released. Exclusive locks are placed when a page is to be modified (such as for an UPDATE, INSERT, or DELETE).

An exclusive lock differs from a shared lock because it does not permit another user to place any type of lock on the same data.

expression—An item or a combination of items and operators that yield a single value. Examples are column names which yield the value of the column in successive rows, arithmetic expressions built with operators such as + or - that yield the result of performing the operation, and functions which yield the value of the function for its argument.

extent page—A database page used when a row is INSERTed that is longer than a page or when a row is UPDATEd and there is not enough space in the original page to hold the data.

external function—A user-defined function that resides in an "external" DLL (Dynamic Link Library) invoked within a SQLBase stored procedure.

event—See timer event.

field—See column.

file server—A computer that allows network users to store and share information.

FLOAT—A column data type that stores floating point numbers.

floating point—A number represented as a number followed by an exponent designator (such as 1.234E2, -5.678E2, or 1.234E-2). Also called E-notation or scientific notation.

foreign key—Foreign keys logically connect different tables. A foreign key is a column or combination of columns in one table whose values match a primary key in another table. A foreign key can also be used to match a primary key within the same table.

front-end—See client.

function—A predefined operation that returns a single value per row in the output result table.

grant—That act of a system administrator to permit a user to make specified use of a database. A user may be granted access to an entire database or specific portions, and have unlimited or strictly-limited power to display, change, add, or delete data.

GUI (Graphical User Interface)—A graphics-based user interface with windows, icons, pull-down menus, a pointer, and a mouse. Microsoft Windows and Presentation Manager are examples of graphical user interfaces.

history file—Contains previous versions of changed database pages. Used when read-only (RO) isolation level is enabled.

host language—A program written in a language that contains SQL commands.

identifier—The name of a database object.

index—A data structure associated with a table used to locate a row without scanning an entire table. An index has an entry for each value found in a table's indexed column or columns, and pointers to rows having that value. An index is logically ordered by the values of a key. Indexes can also enforce uniqueness on the rows in a table.

INTEGER—A column data type that stores a number without a decimal point. Also call INT.

isolation level—The extent to which operations performed by one user can be affected by (are isolated from) operations performed by another user. The isolation levels are Read Repeatability (RR), Cursor Stability (CS), Release Locks (RL), and Read Only (RO).

join—A query that retrieves data from two or more tables. Rows are selected when columns from one table match columns from another table. See also Cartesian product, self-join, equijoin, natural join, theta join, and outer join.

key—A column or a set of columns in an index used to identify a row. A key value can be used to locate a row.

keyword—One of the predefined words in a command language.

local area network (LAN)—A collection of connected computers that share data and resources, and access other networks or remote hosts. Usually, a LAN is geographically confined and microcomputer-based.

lock—To temporarily restrict other users access to data to maintain consistency. Locking prevents data from being modified by more than one user at a time and prevents data from being read while being updated. A lock serializes access to data and prevents simultaneous updates that might result in inconsistent data. See shared lock (S-lock) and exclusive lock (X-lock).

logical operator—A symbol for a logical operation that connects expressions in a WHERE or HAVING clause. Examples are AND, OR, and NOT. An expression formed with logical operators evaluates to either TRUE or FALSE. Logical operators define or limit the information sought. Also called Boolean operator.

LONG VARCHAR—In SQL, a column data type where the value can be longer than 254 bytes. The user does not specify a length. SQLBase stores LONG VARCHAR columns as variable-length strings. Also called LONG.

mathematical function—An operation such as finding the average, minimum, or maximum value of a set of values.

media recovery—Restoring data from backup after events such as a disk head crash, operating system crash, or a user accidentally dropping a database object.

message buffer—The input message buffer is allocated on both the client computer and the database server. The database server builds an input message in this buffer on the database server and sends it across the network to a buffer on the client. It is called an input message buffer because it is input from the client's point of view.

The out put message buffer is allocated on both the client computer and on the database server. The client builds an output message in this buffer and sends it to a buffer on the database server. It is called an output message buffer because it is output from the client's point of view.

modulo—An arithmetic operator that returns an integer remainder after a division operation on two integers.

multi-user—The ability of a computer system to provide its services to more than one user at a time.

natural join—An equijoin where the value of the columns being joined are compared on the basis of equality. All the columns in the tables are included in the results but only one of each pair of joined columns is included.

NDS (NetWare Directory Services)—A network-wide directory included with NetWare 4.x, that provides global access to all network resources, regardless of their physical location. The directory is accessible from multiple points by network users, services and applications.

nested query—See subquery.

NetWare—The networking components sold by Novell. NetWare is a collection of data link drivers, a transport protocol stack, client computer software, and the NetWare server operating system. NetWare runs on Token Ring, Ethernet, and ARCNET.

NetWare 386—A server operating system from Novell for computers that controls system resources on a network.

NLM (NetWare Loadable Module)—An NLM is a NetWare program that you can load into or unload from server memory while the server is running. When loaded, an NLM is part of the NetWare operating system. When unloaded, an NLM releases the memory and resources that were allocated for it.

null—A value that indicates the absence of data. Null is not considered equivalent to zero or to blank. A value of null is not considered to be greater than, less than, or equivalent to any other value, including another value of null.

NUMBER—A column data type that contains a number, with or without a decimal point and a sign.

numeric constant—A fixed value that is a number.

ODBC—The Microsoft Open DataBase Connectivity (ODBC) standard, which is an application programming interface (API) specification written by Microsoft. It calls for all client applications to write to the ODBC standard API and for all database vendors to provide support for it. It then relies on third-party database drivers or access tools that conform to the ODBC specification to translate the ODBC standard API calls generated by the client application into the database vendor’s proprietary API calls.

operator—A symbol or word that represents an operation to be performed on the values on either side of it. Examples of operators are: arithmetic (+, -, *, /), relational (=, !=, >, <, >=, <=), and logical (AND, OR, NOT).

optimization—The determination of the most efficient access strategy for satisfying a database access.

outer join—A join in which both matching and non-matching rows are returned. Each preserved row is joined to an imaginary row in the other table in which all the fields are null.

outer query—When a query is nested within another query, the main query is called the outer query and the inner query is called the subquery. An outer query is executed once for each row selected by the subquery. A subquery cannot be evaluated independently but that depends on the outer query for its results. Also see subquery.

page—The physical unit of disk storage that SQLBase uses to allocate space to tables and indexes.

parent table—The table containing the primary key.

parse—To examine a command to make sure that it is properly formed and that all necessary information is supplied.

partitioning—A method of setting up separate user areas to maximize disk space. Databases can be stretched across several different network partitions.

password—A sequence of characters that must be entered to connect to a database. Associated to each password is an authorization-id.

picture—A string of characters used to format data for display.

precedence—The default order in which operations are performed in an expression.

precision—The maximum number of digits in a column.

precompilation—Processing of a program containing SQL commands or procedures that takes place before compilation. SQL commands are replaced with statements that are recognized by the host language compiler. Output from precompilation includes source code that can be submitted to the compiler.

predicate—An element in a search condition that expresses a comparison operation that states a set of criteria for the data to be returned by a query.

primary key—The columns or set of columns that are used to uniquely identify each row in a table. All values for a key are unique and non-null.

privilege—A capability given to a user to perform an action.

procedure—A named set of SAL or SQL statements that can contain flow control language. You compile a procedure for immediate and/or later execution.

query—A request for information from a database, optionally based on specific conditions. For example, a request to list all customers whose balance is greater than $1000. Queries are issued with the SELECT command.

Read Only (RO)—The isolation level where pages are not locked, and no user has to wait. This gives the user a snapshot view of the database at the instant that the transaction began. Data cannot be updated while in the read-only isolation level.

Read Repeatability (RR)—The isolation level where if data is read again during a transaction, it is guaranteed that those rows would not have changed. Rows referenced by the program cannot be changed by other programs until the program reaches a commit point. Subsequent queries return a consistent set of results (as though changes to the data were suspended until all the queries finished). Other users will not be able to update any pages that have been read by the transaction. All shared locks and all exclusive locks are retained on a page until the transaction completes. Read repeatability provides maximum protection from other active application programs. This ensures a high level of consistency, but lowers concurrency. SQLBase default isolation level.

REAL—A column data type that stores a single-precision number.

record—See row.

recovery—Rebuilding a database after a system failure.

referential cycle—Tables which are dependents of one another.

referential integrity—Guarantees that all references from one database table to another are valid and accurate. Referential integrity prevents problems that occur because of changes in one table which are not reflected in another.

relation—See table.

relational database—A database that is organized and accessed according to relationships between data items. A relational database is perceived by users as a collection of tables.

relational operator—A symbol (such as =, >, or <) used to compare two values. Also called comparison operator.

Release Locks (RL)—With the Cursor Stability isolation level, when a reader moves off a database page, the shared lock is dropped. However, if a row from the page is still in the message buffer, the page is still locked.

In contrast, the Release Lock (RL) isolation level increases concurrency. By the time control returns to the application, all shared locks have been released.

repeating query—See correlated subquery.

requester—See client.

restore—Copying a backup of a database or its log files to a database directory.

restriction mode—In restriction mode, the result set of one query is the basis for the next query.   Each query further restricts the result set. This continues for each subsequent query.

result set mode—Normally, result table rows are displayed and scrolled off the screen. In result set mode, the rows of the result table are available for subsequent scrolling and retrieval.

result table—The set of rows retrieved from one or more tables or views during a query. A cursor allows the rows to be retrieved one by one.

revoke—The act of withdrawing a user's permission to access a database.

rollback—To restore a database to the condition it was in at its last COMMIT. A ROLLBACK cancels a transaction and undoes any changes that it made to the database. All locks are freed unless cursor-context preservation is on.

rollforward—Reapplying changes to a database. The transaction log contains the entries used for rollforward.

router—A client application talks to a SQLBase server through a router program. The router enables a logical connection between a client and the server. Once this connection is established on the LAN, the client application uses the router program to send SQL requests to the server and to receive the results.

row—A set of related columns that describe a specific entity. For example, a row could contain a name, address, telephone number. Sometimes called record or tuple.

ROWID—A hidden column associated with each row in a SQLBase table that is an internal identifier for the row. The ROWID can be retrieved like any other column.

ROWID validation—A programming technique that ensures that a given row that was SELECTed has not been changed or deleted by another user during a session. When a row is updated, the ROWID is changed.

SAP (Service Advertisement Protocol)—A NetWare protocol that resources (such as database servers) use to publicize their services and addresses on a network.

savepoint—An intermediate point within a transaction to which a user can later ROLLBACK to cancel any subsequent commands, or COMMIT to complete the commands.

scale—The number of digits to the right of the decimal point in a number.

search condition—A criterion for selecting rows from a table. A search condition appears in a WHERE clause and contains one or more predicates.

search—To scan one or more columns in a row to find rows that have a certain property.

self-join—A join of a table with itself. The user assigns the two different correlation names to the table that are used to qualify the column names in the rest of the query.

self-referencing table—A table that has foreign and primary keys with matching values within the same table.

server—A computer on a network that provides services and facilities to client applications.

shared lock (S-lock)—A shared lock permits other users to read data, but not to change it. A shared lock lets users read data concurrently, but does not let a user acquire an exclusive lock on the data until all the users' shared locks have been released. A shared lock is placed on a page when the page is read (during a SELECT). At a given time, more than one user can have a shared lock placed on a page. The timing of the release of a shared lock depends on the isolation level.

A shared lock differs from an exclusive lock because it permits more than one user to place a lock on the same data.

single-user—A computer system that can only provide its services to one user at a time.

SMALLINT—A column data type that stores numbers without decimal points.

socket—An identifier that Novell's IPX (Internetwork Packet Exchange) uses to route packets to a specific program.

SPX (Sequenced Packet Exchange)—A Novell communication protocol that monitors network transmissions to ensure successful delivery. SPX runs on top of Novell's IPX (Internetwork Packet Exchange).

SQL (Structured Query Language)—A standard set of commands used to manage information stored in a database. These commands let users retrieve, add, update, or delete data. There are four types of SQL commands: Data Definition Language (DDL), Data Manipulation Language (DML), Data Query Language (DQL), and Data Control Language (DCL). SQL commands can be used interactively or they can be embedded within an application program. Pronounced ess-que-ell or sequel.

SQLBase—A relational DBMS that lets users access, create, and update data.

SQLTalk—SQLTalk is an interactive user interface for SQLBase that is used to manage a relational database. SQLTalk has a complete implementation of SQL and many extensions. SQLTalk is a client application.

static SQL—See embedded SQL.

statistics—Attributes about tables such as the number of rows or the number of pages. Statistics are used during optimization to determine the access path to a table.

storage group—A list of database areas. Storage groups provide a means to allow databases or tables to be stored on different volumes.

stored procedure—A precompiled procedure that is stored on the backend for future execution.

string delimiter—A symbol used to enclose a string constant. The symbol is the single quote (').

string—A sequence of characters treated as a unit of data.

subquery—A SELECT command nested within the WHERE or HAVING clause of another SQL command. A subquery can be used anywhere an expression is allowed if the subquery returns a single value. Sometimes called a nested query. Also called subselect. See also correlated subquery.

synonym—A name assigned to a table, view, external function that may be then used to refer to it. If you have access to another user's table, you may create a synonym for it and refer to it by the synonym alone without entering the user's name as a qualifier.

syntax—The rules governing the structure of a command.

system catalog—A set of tables SQLBase uses to store metadata. System catalog tables contain information about database objects, privileges, events, and users. Also called data dictionary.

system keywords—Keywords that can be used to retrieve system information in commands.

table—The basic data storage structure in a relational database. A table is a two-dimensional arrangement of columns and rows. Each row contains the same set of data items (columns). Sometimes called a relation.

table scan—A method of data retrieval where a DBMS directly searches all rows in a table sequentially instead of using an index.

theta join—A join that uses relational operators to specify the join condition.

TIME—A column data type in the form of a value that designates a time of day in hours, minutes, and possibly seconds (a two- or three-part value).

timeout—A time interval allotted for an operation to occur.

TIMESTAMP—A column data type with a seven-part value that designates a date and time. The seven parts are year, month, day, hour, minutes, seconds, and microseconds (optional). The format is:

```
yyyy-mm-dd-hh.mm.ss.nnnnnn
```

timer event—Executes a procedure at a predetermined time. You can optionally repeat the timer event at specified intervals.

token—A character string in a specific format that has some defined significance in a SQL command.

Token-Ring—A LAN with ring topology (cable connected at the ends). A special data packet called a token is passed from one computer to another. When a computer gets the token, it can attach data to it and transmit. Each computer passes on the data until it arrives at its destination. The receiver marks the message as being received and sends the message on to the next computer. The message continues around the ring until the sender receives it and frees the token.

tokenized error message—An error message formatted with tokens in order to provide users with more informational error messages. A tokenized error message contains one or more variables that SQLBase substitutes with object names (tokens) when it returns the error message to the user.

transaction—A logically-related sequence of SQL commands that accomplishes a particular result for an application. SQLBase ensures the consistency of data by verifying that either all the data changes made during a transaction are performed, or that none of them are performed. A transaction begins when the application starts or when a COMMIT or ROLLBACK is executed. The transaction ends when the next COMMIT or ROLLBACK is executed. Also called logical unit of work.

transaction log—A collection of information describing the sequence of events that occur while running SQLBase. The information is used for recovery if there is a system failure. A log includes records of changes made to a database. A transaction log in SQLBase contains the data needed to perform rollbacks, crash recovery, and media recovery.

trigger—Activates a stored procedure that SQLBase automatically executes when a user attempts to change the data in a table, such as on a DELETE or UPDATE command.

two-phase commit—The protocol that coordinates a distributed transaction commit process on all participating databases.

tuple—See row.

unique key—One or more columns that must be unique for each row of the table. An index that ensures that no identical key values are stored in a table.

username—See authorization-id.

value—Data assigned to a column, a constant, a variable, or an argument.

VARCHAR—See CHAR.

variable—A data item that can assume any of a given set of values.

view—A logical representation of data from one or more base tables. A view can include some or all of the columns in the table or tables on which it is defined. A view represents a portion of data generated by a query. A view is derived from a base table or base tables but has no storage of its own. Data for a view can be updated in the same manner as for a base table. Sometimes called a virtual table.

wildcard—Characters used in the LIKE predicate that can stand for any one character (the underscore _) or any number of characters (the percent sign %) in pattern-matching.

Windows—A graphical user interface from Microsoft that runs under DOS.

With Windows, commands are organized in lists called menus. Icons (small pictures) on the screen represent applications. A user selects a menu item or an icon by pointing to it with a mouse and clicking.

Applications run in windows that can be resized and relocated. A user can run two or more applications at the same time and can switch between them. A user can run multiple copies of the same application at the same time.

write-ahead log (WAL)—A transaction logging technique where transactions are recorded in a disk-based log before they are recorded in the physical database. This ensures that active transactions can be rolled back if there is a system crash.

Index