

# SQLBase

## *SQLTalk Command Reference*

20-2108-1004



# Trademarks

Gupta, the Gupta logo, Gupta Powered, the Gupta Powered logo, Centura, the Centura logo, Centura Web Developer, Fast Facts, Object Nationalizer, Quest, QuickObjects, SQL/API, SQLBase, SQLConsole, SQLGateway, SQLHost, SQLNetwork, SQLRouter, SQLTalk, and Team Object Manager are trademarks of Gupta Technologies LLC and may be registered in the United States of America and/or other countries. The trademarks TeamWindows, ReportWindows and EditWindows, and the registered trademark SQL Windows, are all exclusively used and licensed by Gupta Technologies LLC.

Adobe is a trademark of Adobe Systems, Incorporated.

IBM, OS/2, NetBIOS, and AIX are registered trademarks of International Business Machines Corporation.

Java, JavaScript, and Solaris are trademarks of Sun Microsystems, Incorporated.

Microsoft, Outlook, PowerPoint, Visual C++, Visual Studio, Internet Explorer, Internet Information Server, DOS, Windows, ActiveX, MSDN, SQL Server, and Visual Basic are either registered trademarks or trademarks of Microsoft Corporation in the United States of America and/or other countries.

Netscape FastTrack and Navigator are trademarks of Netscape Communications Corporation.

Novell is a registered trademark, and NetWare is a trademark of Novell, Incorporated.

All other product or service names mentioned herein are trademarks or registered trademarks of their respective owners.

# Copyright

Copyright © 2003 by Gupta Technologies LLC. All rights reserved.

*SQLTalk Command Reference*

20-2104-1004

August 2003

# Preface

---

This is a reference manual for SQLTalk, an interactive user interface that allows you to enter SQL commands. SQLTalk can be used as an interface for other databases besides SQLBase, such as DB2.

## Who should read this manual

The *SQLTalk Command Reference* is written for anyone using the advanced features of SQLTalk. This includes:

- **Application developers** build client applications that access databases using frontend Gupta products like SQLTalk, Team Developer, and the SQL/API.
- **Database Administrators** perform day-to-day operation and maintenance of the database. They design the database, create database objects, load data, control access, perform backup and recovery, and monitor performance.
- **End users** use SQL to query and change data.

This manual assumes you have:

- Knowledge of relational databases and SQL.

---

**Note:** This manual is not intended to be a SQL tutorial.

---

---

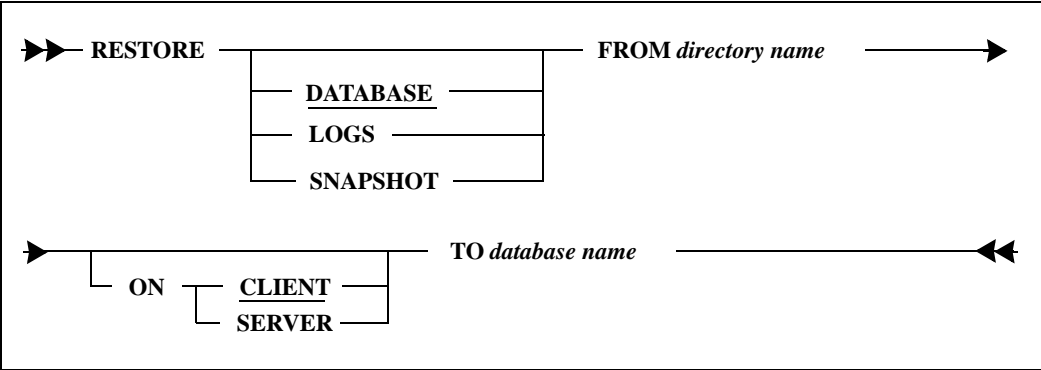
# Summary of chapters

This manual is organized in the chapters in the table below.

1	Introduction to SQLTalk	Provides an overview of SQLTalk.
2	SQLTalk Command Reference	Describes each SQLTalk command. Arranged alphabetically.
3	SQLTalk Reserved Words	This chapter lists SQLTalk reserved words.

# Syntax diagrams

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



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


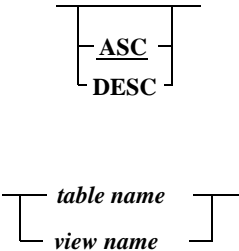
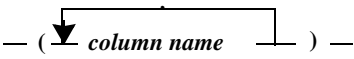
The line with the command name (RESTORE) is the main line of the command. Mandatory keywords and arguments (such as FROM *directory name*) appear on the main line or a continuation of the main line.

The above diagram could generate the commands shown in these examples:

```
RESTORE DATABASE FROM \DEMOBKUP TO DEMO;  
RESTORE LOGS FROM \ACCTBKUP ON SERVER TO ACCTREC;
```

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.
	<p>An optional item with alternate 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.</p> <p>If an item is mandatory, the first alternative is on the main line (this example is from the UPDATE command).</p>
	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

The table below show the notation conventions that this manual uses.

Notation	Explanation
You	A developer who reads this manual
User	The end-user of applications that you write
<b>bold</b> type	Menu items, push buttons, and field names. Things that you select. Keyboard keys that you press.
Courier 9	SQLWindows/32 or C language code example
SQL.INI MAPDLL.EXE	Program names and file names
Precaution	Warning:
Vital information	Important:
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



***Gupta Books Online.*** The Gupta 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 Gupta 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.

***Internet.*** Gupta's world wide Web site contains information about Gupta Technologies LLC's partners, products, sales, support, training, and users. The URL is <http://www.guptaworldwide.com>.

The technical services 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 Gupta's online newsgroups, links to shareware tools, product bulletins, white papers, and downloadable product updates.

Our Web site also includes information on training, including course descriptions, class schedules, and certified training partners.

## 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  
Gupta Technologies LLC  
975 Island Drive  
Redwood Shores, CA 94065

or send email, with comments or suggestions to:

[techpubs@guptaworldwide.com](mailto:techpubs@guptaworldwide.com)

# Contents

---

<b>Preface</b> .....	1-3
Who should read this manual .....	1-4
Summary of chapters .....	1-5
Syntax diagrams .....	1-6
Notation conventions .....	1-8
Other helpful resources .....	1-9
Send comments to.....	1-10
<b>Introduction to SQLTalk</b> .....	1-1
What is SQLTalk? .....	1-2
SQLTalk command categories .....	1-2
Session control .....	1-3
Database administration.....	1-3
Report writing .....	1-3
Stored commands and procedures.....	1-4
Command files .....	1-4
Precompiled commands .....	1-4
Environment control .....	1-4
Editing commands .....	1-5
SQL commands .....	1-5
Starting the SQLTalk program .....	1-6
Database name, user name, and password....	1-10
Entering SQLTalk commands.....	1-10
SQLTalk run options .....	1-11
Initialization file .....	1-11
NOCONNECT option .....	1-11
Batch option (BAT) .....	1-12

---

Examples .....	1-13
Result set mode .....	1-13
Cursors .....	1-15
Restriction mode .....	1-16
Bind variables .....	1-19
Examples of bind variables .....	1-20
Entering long data with bind variables .....	1-21
Error messages .....	1-23
About error.sql .....	1-23
About message.sql .....	1-23
Displaying errors .....	1-24
Setting the error level .....	1-24
<b>SQLTalk Command Reference .....</b>	<b>2-1</b>
SQLTalk command summary .....	2-2
ALTER COMMAND .....	2-4
ALTER DBSECURITY .....	2-5
ALTER EXPORTKEY .....	2-9
BACKUP .....	2-10
BEGIN CONNECTION .....	2-14
BREAK .....	2-16
BTITLE .....	2-17
COLUMN .....	2-19
COMPUTE .....	2-26
CONNECT .....	2-30
COPY .....	2-33
DBERROR .....	2-34
DISCONNECT .....	2-35
END CONNECTION .....	2-37
ERASE .....	2-38
ERASE FILTER .....	2-39
EXECUTE .....	2-40
EXIT .....	2-42

---

FETCH .....	2-43
LEFT .....	2-44
LIST .....	2-45
PAUSE .....	2-46
PERFORM .....	2-46
PREPARE .....	2-47
PRINT .....	2-49
RELEASE LOG .....	2-50
REMARK .....	2-51
REORGANIZE .....	2-51
REORGANIZE with encrypted databases .....	2-53
RESTORE .....	2-53
RETRIEVE .....	2-57
RIGHT .....	2-59
ROLLFORWARD .....	2-61
RUN .....	2-63
SAVE .....	2-67
SAVE FILTER .....	2-68
SET .....	2-69
SET SERVER .....	2-101
SET SPOOL .....	2-102
SHOW .....	2-104
SHOW CONNECTION .....	2-106
SHOW DATABASES .....	2-107
SHUTDOWN .....	2-108
SQLGET .....	2-109
SQLSET .....	2-111
STORE .....	2-113
TTITLE .....	2-115
UNDO .....	2-117
USE .....	2-118
SQLTalk Reserved Words .....	3-1

---

SQLTalk reserved words. . . . . 3-2

**Glossary** . . . . . Glossary-1

# Chapter 1

## Introduction to SQLTalk

---

This chapter provides an overview of SQLTalk and includes the following information:

- The purpose of SQLTalk
- The SQLTalk command categories
- How to start SQLTalk
- SQLTalk run options
- How to use result set mode to browse
- How bind variables can supply new or changed data
- Error handling in SQLTalk

# What is SQLTalk?

SQLTalk is an interactive user interface for SQLBase. SQLBase is a relational database management system (RDBMS), providing complete implementation of Structured Query Language (SQL) as well as its own control language. It is designed and built specifically for PC networks supporting various LAN/WAN configurations.

Besides accessing SQLTalk through its user interface, you can also access SQLTalk as a custom control available through Gupta's SQLConsole or Team Builder.

With SQLTalk, you can perform these tasks:

- Define the structure of a database
- Add, delete, and change data in a database
- Query a database
- Execute batch and interactive SQLTalk scripts
- Edit SQLTalk scripts
- Control security and access for a database
- Generate reports
- Test SQL commands before they are embedded in an application program
- Perform DBA functions

SQLTalk can also be used as an interface for other databases besides SQLBase, such as DB2.

## SQLTalk command categories

SQLTalk commands can be grouped into the following categories:

- Session control
- Database administration
- Report writing
- Stored commands and procedures
- Command files
- Precompiled commands
- Environment control
- SQL commands

## Session control

These commands connect or disconnect cursors, databases, and connection handles.

BEGIN CONNECTION  
CONNECT  
DISCONNECT  
END CONNECTION  
EXIT  
USE

## Database administration

These commands perform database administration functions such as backup and restore.

ALTER DBSECURITY  
ALTER EXPORTKEY  
BACKUP  
BACKUP  
COPY  
FETCH  
RELEASE LOG  
REORGANIZE  
RESTORE  
ROLLFORWARD  
SET SERVER  
SHUTDOWN

## Report writing

These commands format, display, and print the results of a SQL query in a complex multi-page report.

BREAK  
BTITLE  
COLUMN  
COMPUTE  
LEFT  
PRINT  
RIGHT  
TTITLE

## Stored commands and procedures

These commands store, retrieve, run, and erase SQL stored commands and procedures.

```
ALTER COMMAND
ERASE
EXECUTE
RETRIEVE
STORE
```

These SQL commands grant and revoke privileges to users on stored procedures.

```
GRANT EXECUTE ON
REVOKE EXECUTE ON
```

## Command files

These commands let you store often-used commands in a file that can be executed.

```
PAUSE
REMARK
RUN
SAVE
```

## Precompiled commands

These commands compile and execute SQL commands:

```
PREPARE
PERFORM
```

## Environment control

The SET command lets you control the environment for the SQLTalk session. The SHOW command displays the current settings of the environment. The DBERROR command displays the message text, reason, and remedy for a specific error.

```
SET
SET SPOOL
SHOW
SHOW CONNECTION
SHOW DATABASES
DBERROR
```

## Editing commands

These commands let you edit command input.

EDIT  
LIST  
UNDO

## SQL commands

These SQL commands can be executed from the SQLTalk command prompt.

ALTER DATABASE  
ALTER DBAREA  
ALTER EXTERNAL FUNCTION  
ALTER PASSWORD  
ALTER STOGROUP  
ALTER TABLE  
ALTER TABLE (error messages)  
ALTER TABLE (referential integrity)  
ALTER TRIGGER  
AUDIT MESSAGE  
CHECK DATABASE  
CHECK INDEX  
CHECK TABLE  
COMMENT ON  
COMMIT  
CREATE DATABASE  
CREATE DBAREA  
CREATE EXTERNAL FUNCTION  
CREATE INDEX  
CREATE STOGROUP  
CREATE SYNONYM  
CREATE TABLE  
CREATE TRIGGER  
CREATE VIEW  
DBATTRIBUTE  
DEINSTALL DATABASE  
DELETE  
DROP DATABASE  
DROP DBAREA  
DROP EXTERNAL FUNCTION  
DROP INDEX  
DROP STOGROUP  
DROP SYNONYM

DROP TABLE  
DROP TRIGGER  
DROP VIEW  
GRANT (database authority)  
GRANT (table privileges)  
GRANT EXECUTE ON  
INSERT  
INSTALL DATABASE  
LABEL  
LOAD  
LOCK DATABASE  
PROCEDURE  
REVOKE (database authority)  
REVOKE (table privileges)  
REVOKE EXECUTE ON  
ROLLBACK  
ROWCOUNT  
SAVEPOINT  
SELECT  
SET DEFAULT STOGROUP  
START AUDIT  
STOP AUDIT  
UNLOAD  
UNION  
UNLOCK DATABASE  
UPDATE  
UPDATE STATISTICS

## Starting the SQLTalk program

This section describes how to start SQLTalk on each client platform.

If you are not using a single-user engine, you may need to start the SQLBase server on the server machine before running SQLTalk on the client machine. Read either the *SQLBase Starter Guide* or the *Database Administrator's Guide* for information on starting the server from your particular platform.

The clients that can connect to your SQLBase server vary according to your platform. In addition, some SQLBase servers and clients support multiple network communication protocols. Read the *SQLBase Starter Guide* for details.

The name of the Windows SQLTalk program is *sqltalk.exe*.

## Starting sqltalk.exe from Windows

1. Start a supported version of Windows (Windows 98, ME, NT, 2000, Server 2003, and XP).
2. Do one of the following:
  - Click on the SQLTalk icon,  
or
  - Choose **Start, Programs, Gupta, SQLBase 8.5** (or similar Gupta product), **SQLTalk**.
  - Choose **File, Run** or **Start, Run** in the Windows Program Manager. Enter `c:\Gupta\sqltalk.exe` in the Run dialog box and click **OK**. (Your path might differ from `c:\Gupta`.)

If you choose the last method, you can enter optional command-line arguments. You can specify the name of a script file containing SQLTalk commands, which will be run automatically when SQLTalk starts. You can also specify the name of a configuration file which will supply all client connectivity information (such as server names and available network protocols) to SQLTalk. The example below shows both such arguments in use.

```
SQLTalk.exe "c:\program files\km.sql"
           "ini=c:\program files\sql.ini"
```

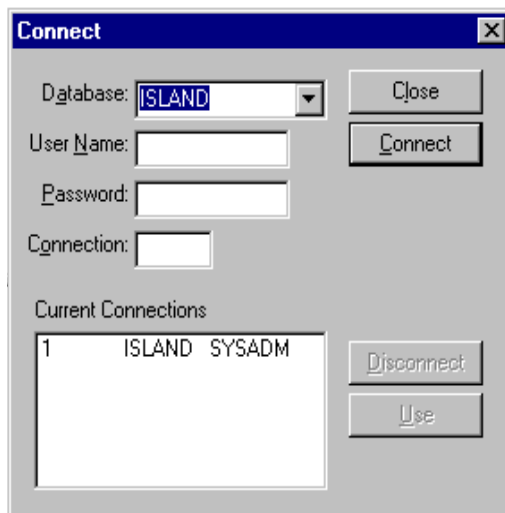
Both of these arguments are optional. If you do not use the configuration file argument, SQLTalk presumes that the configuration file name is `SQL.INI`, and searches for it using an algorithm described in chapter 3 of the *Database Administrator's Guide*.

## Running SQLTalk

Once SQLTalk has started, here are the steps to begin using it:

1. **Enter a database.**
  - Select **Connect** from the Session menu.

- In the dialog box, enter **ISLAND** for the database and click **Connect**.



The Current Connections box displays ISLAND as the database name, which confirms your connection to this database.

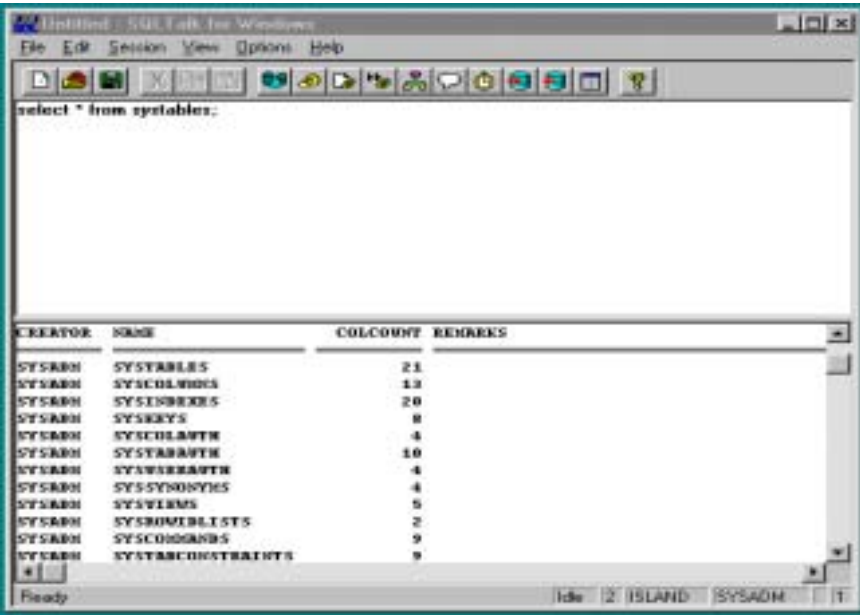
---

**Note:** The *username* and *password* defaults to SYSADM for all new databases until you define another user name and password from within SQLTalk. Once you have created one or more additional user names, you can override the default by entering another name.

---

## 2. Start entering commands.

You can start entering commands at the cursor in the top pane of the SQLTalk window. For example:



You can enter SQL and SQLTalk commands as described in the *SQL Language Reference Manual* and in this manual. For rules on entering commands, read *Entering SQLTalk commands* in this chapter.

**Note:** You can also connect to a database by entering the following command in SQLTalk's window. For example:

```
CONNECT ISLAND1;  
  
CURSOR 1 CONNECTED TO ISLAND1
```

**3. End SQLTalk.**

When you are ready to end your SQLTalk session, close the application window using the File, Exit menu items, or enter this command at the cursor:

```
exit;
```

## Database name, user name, and password

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.

## Entering SQLTalk commands

Once you have started SQLTalk, you can issue SQL commands or SQLTalk commands.

A SQL command consists of:

- Keywords (such as command names and clause names)
- Arguments
- Operators (such as +, -, \*, /, >, or =)
- Delimiters (such as commas or parentheses)

To execute a SQL or SQLTalk command, either right-click on the Execute Script button with the mouse, or press CONTROL-ENTER or SHIFT F2 while the cursor is within the command. You can also use the Execute Command option from the SQLTalk toolbar or SESSION/EXECUTE COMMAND menu option.

## Command guidelines

- You can have any amount of white space between items.
- A delimiter must mark the end of a command. The delimiters for a command are:
  - A semicolon (;) at the end of the last line.
  - A forward slash (/) at the beginning of a new line after the last line.
- If you enter a *single* command delimiter character (; or /), SQLBase runs the most recently entered command.
- SQLTalk commands can span multiple lines.
- Commands are case insensitive.
- String constants are case sensitive.
- Delimited identifiers (names) are case sensitive.
- String constants usually must be specified within *single* quotes (' ').

## Backslash continuation character ( \ )

You can use a continuation character while entering data or commands. The continuation character is the backslash (\). The continuation character works anywhere except in the first space of a line. The continuation character can be used in unload files.

## Marking input data

SQLTalk commands that contain data values use the backslash (\) to mark the beginning of input data. The end of the data is marked with a forward slash (/) character on a new line followed by a carriage return.

# SQLTalk run options

This section describes options you can use when running SQLTalk.

## Initialization file

SQLTalk can read a command file upon startup. To create this file, use the SQLTalk executable name with the *.tlk* extension.

Any command in this file is executed upon startup, but no output will be displayed on the screen. For example, you can create a command file on the Windows 98 and Windows NT platform called *sqlntltlk.tlk* that contains SQLTalk page set commands (SET<option>). Any command in this file is executed upon startup, but no output is displayed on the screen.

## NOCONNECT option

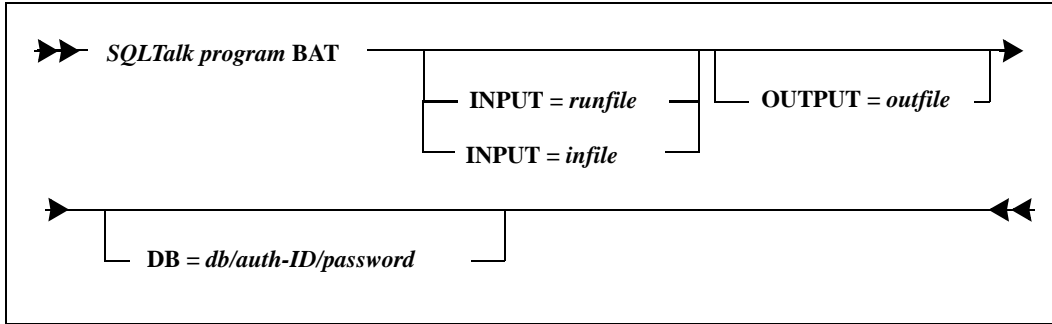
This option is only useful in the batch mode when running SQLTalk under any Windows platform. You cannot be connected to a database to restore it (read the explanation for the RESTORE command). If you want to connect to a server without connecting to a specific database, specify the NOCONNECT option on the command line. For example, enter this command on the Windows NT platform:

```
C:\Gupta> SQLTALK BAT NOCONNECT
```

If you are using a multi-user server, after SQLTalk starts, issue a SET SERVER command to connect to a server. Then include any commands that you require in your batch file.

## Batch option (BAT)

You can run SQLTalk from a command file. BAT is a command line option that enables you to run report and command scripts as batch files. You run the BAT option at the operating system prompt.



## Clauses

You can specify the command line options in any order.

### SQLTalk program BAT

Enter the name of your platform-specific SQLTalk program, followed by BAT. For example, enter this command on the Windows 95 or Windows NT platform:

```
C:\Gupta> SQLTALK BAT
```

### INPUT = runfile

This is a SQLTalk command file to be executed. You create this file with an online editor. For more information on command files, see the RUN command.

### INPUT = infile

All input comes from this file and the SQLTalk command prompt (SQL>) is not displayed. If this clause is not specified, SQLTalk takes command input from the keyboard.

### OUTPUT=outfile

This specifies a file to write all standard output. If this clause is not specified, output is written to the standard output device (usually the display screen).

### DB=db/auth-ID/password

This is the database name, username, and password used for connecting to SQLBase. The system uses the default for any of the three that you do not

supply. For example, if you specify only the password, SQLBase uses the default username and database name.

## Examples

The following examples using the Windows NT SQLTalk program get their input from a command file called *sales.rpt*.

This example writes output to the screen on the Windows NT platform:

```
C:\Gupta> SQLTALK BAT INPUT=SALES.RPT
DB=SALES/WILLIE/XYZ
```

This example writes output to a file called *sales.lst*:

```
C:\Gupta> SQLTALK BAT INPUT=SALES.RPT
OUTPUT=SALES.LST DB=SALES
```

## Result set mode

In a SQL SELECT command, you specify tables, columns, and rows from which to select data.

The database finds the data that matches this specification and displays it in a temporary table called a *result table*. The rows in the result table represent the data that either meet the conditions or have undergone the operations specified in the query. If no data qualifies, the result table contains zero rows.

For example, the following query selects the employee name and number from the ENGINEERS table:

```
SELECT EMPL_NUM, NAME FROM ENGINEERS;
```

<u>EMPL_NUM</u>	<u>NAME</u>
100	Paul Atkins
104	Bob Smith3
107	Murray Rochester
102	Larry Sanchez
101	Sheila Brown
106	Sam Valdez
105	Rob Jones
103	Anna Rice
108	Mary Adams
109	Nancy Bonet
110	Richard Park
111	Dan Chester

Result table from  
EMPL\_NUM query

Normally, SQLBase displays result table rows, and then rolls them off the screen. However, in *result set mode* (also called scroll mode), the rows of the result table are available for subsequent scrolling and retrieval. This is similar to the SCROLL CURSOR capability in the ANSI SQL standard.

While in result set mode, you can:

- Starting at any row position, scroll through the result table.
- INSERT, UPDATE, or DELETE rows in the result table.

Result set mode is useful for a *browsing* application where you need to examine data before deciding what to do with it.

Turn on result set mode with the SET SCROLL ON command. Result set mode is disabled by default.

After a query, the database is positioned at the first row of the result set. When you issue a FETCH command, the database returns the number of rows specified starting at the first row. Each subsequent FETCH command retrieves the next row, until the end of the result set.

Before running FETCH, you can set the starting position within the result set with the SET SCROLLROW command.

This section shows how to use result set mode. It also introduces the concept of cursors and restriction modes, which are described in more detail in the sections Cursors and Restriction mode on page 16.

## Using result set mode

1. Enable result set mode:

```
SET SCROLL ON;
```

2. Optionally, turn on restriction mode:

```
SET RESTRICTION ON;
```

3. Issue the SELECT command:

```
SELECT ...;
```

4. Assign a name to the current cursor.

```
SET CURSORNAME char-string;
```

The cursor name can be used in the INSERT, UPDATE, and DELETE commands.

5. Set the position within the result table:

```
SET SCROLLROW integer-constant;
```

where `integer-constant` is the starting row number. Row number 0 is the first row.

6. Display a part of the result table:

```
FETCH integer-constant;
```

where ***integer-constant*** is the number of rows to display. The display starts at the row number specified by the previous `SET SCROLLROW` command.

7. Connect to a different cursor. This is necessary so that the result set for the first cursor is not invalidated after the operation.

```
CONNECT 2;
```

8. Execute one of the following:

- An `UPDATE` command with the `WHERE CURRENT OF` clause.
- A `DELETE` command with the `WHERE CURRENT OF` clause.
- An `INSERT` command with the `ADJUSTING` clause.

9. Go back to the first cursor which still has a valid result set:

```
USE 1;
```

## Cursors

SQLBase uses *cursors* to mark the row position in a result set. Once a result set has been created, an application can position the cursor on any row in the result set. Once the cursor is positioned, subsequent fetches start from that position.

### Modifying data

While in result set mode, you can change or add data with the following commands:

- An `UPDATE` or `DELETE` command with the `WHERE CURRENT OF` clause.
- An `INSERT` command with the `ADJUSTING` clause.

These are called *cursor-controlled* operations because they rely on the cursor specified in the `WHERE CURRENT OF` or `ADJUSTING` clause.

### Using multiple cursors

An application can use multiple SQL cursors. Here are some examples:

- The first cursor (for a `SELECT` command) keeps track of the "current" cursor position in a result table while a second cursor is used to execute an `UPDATE` or `DELETE`.
- An application can update a column in one table based on a value in another table. A separate cursor is associated with each table.

- An application can access two databases simultaneously through multiple cursors. Each database maintains its own transaction and rollback recovery independently.

Use the SQLTalk `USE`, `CONNECT`, and `SET CURSORNAME` commands to perform these operations.

## Restriction mode

While in result set mode, *restriction mode* can filter a query’s result set to form the basis for successive queries from that table or tables. Each query restricts the result set further, until you obtain the desired result. This is useful for browsing applications because you can continue to narrow the focus of a query.

Use `SET RESTRICTION ON` to turn on restriction mode. You must precede this command with a `SET SCROLL ON` command.

You can revert to the most recent result set with the `UNDO` command.

This example uses the data from the `ENGINEERS` table:

**SELECT \* FROM ENGINEERS ;**

EMPL_NUM	NAME	REP_OFFICE	TITLE	HIRE_DATE	MANAGER
100	Paul Atkins	10	Manager	1988-02-12	
104	Bob Smith	20	Sen. Engineer	1992-09-05	103
107	Murray Rochester	30	Sen. Engineer	1991-01-25	106
102	Larry Sanchez	10	Sen. Engineer	1989-06-12	100
101	Sheila Brown	10	Engineer	1990-10-10	100
106	Sam Valdez	30	Manager	1990-04-20	
105	Rob Jones	20	Engineer	1991-09-08	103
103	Anna Rice	20	Manager	1985-07-10	
108	Mary Adams	40	Manager	1988-08-10	
109	Nancy Bonet	40	Sen. Engineer	1989-11-12	108
110	Richard Park	40	Engineer	1990-11-14	108
111	Dan Chester	40	Engineer	1987-03-22	111

1. First, turn on result set mode:

SET SCROLL ON;
2. Turn on restriction mode:

SET RESTRICTION ON;
3. The next query refines the query to employees whose employee number is greater than 102:

```
SELECT * FROM ENGINEERS WHERE EMPL_NUM>102;
```

EMPL_NUM	NAME	REP_OFFICE	TITLE	HIRE_DATE	MANAGER
104	Bob Smith	20	Sen. Engineer	1992-09-05	103
107	Murray Rochester	30	Sen. Engineer	1991-01-25	106
106	Sam Valdez	30	Manager	1990-04-20	
105	Rob Jones	20	Engineer	1991-09-08	103
103	Anna Rice	20	Manager	1985-07-10	
108	Mary Adams	40	Manager	1988-08-10	
109	Nancy Bonet	40	Sen. Engineer	1989-11-12	108
110	Richard Park	40	Engineer	1990-11-14	108
111	Dan Chester	40	Engineer	1987-03-22	111

4. The next query refines the query to employees whose office number is less than 40:

```
SELECT * FROM ENGINEERS WHERE REP_OFFICE <40;
```

EMPL_NUM	NAME	REP_OFFICE	TITLE	HIRE_DATE	MANAGER
104	Bob Smith	20	Sen. Engineer	1992-09-05	103
107	Murray Rochester	30	Sen. Engineer	1991-01-25	106
106	Sam Valdez	30	Manager	1990-04-20	
105	Rob Jones	20	Engineer	1991-09-08	103
103	Anna Rice	20	Manager	1985-07-10	

5. Now, the original unrestricted SELECT statement returns the same result set as the previous SELECT in step 4.

```
SELECT * FROM ENGINEERS;
```

EMPL_NUM	NAME	REP_OFFICE	TITLE	HIRE_DATE	MANAGER
104	Bob Smith	20	Sen. Engineer	1992-09-05	103
107	Murray Rochester	30	Sen. Engineer	1991-01-25	106
106	Sam Valdez	30	Manager	1990-04-20	
105	Rob Jones	20	Engineer	1991-09-08	103
103	Anna Rice	20	Manager	1985-07-10	

6. The following command accesses a different table, which forces SQLBase to create a new result set. The previous result set is lost.

**SELECT \* FROM SERV\_CALLS;**

CALL_NUM	CALL_DATE	CUST	REP	MFR	PRODUCT
2133	1993-05-10	1000	101	ACR	102
6253	1993-05-02	3000	102	LMA	4516
7111	1993-05-09	1001	106	MRP	600
4250	1993-05-14	1050	105	MRP	600

7. Now, when you rerun the SELECT command against the ENGINEERS table, the original result set from step 1 is returned.

**SELECT \* FROM ENGINEERS;**

EMPL_NUM	NAME	REP_OFFICE	TITLE	HIRE_DATE	MANAGER
100	Paul Atkins	10	Manager	1988-02-12	
104	Bob Smith	20	Sen. Engineer	1992-09-05	103
107	Murray Rochester	30	Sen. Engineer	1991-01-25	106
102	Larry Sanchez	10	Sen. Engineer	1989-06-12	100
101	Sheila Brown	10	Engineer	1990-10-10	100
106	Sam Valdez	30	Manager	1990-04-20	
105	Rob Jones	20	Engineer	1991-09-08	103
103	Anna Rice	20	Manager	1985-07-10	
108	Mary Adams	40	Manager	1988-08-10	
109	Nancy Bonet	40	Sen. Engineer	1989-11-12	108
110	Richard Park	40	Engineer	1990-11-14	108
111	Dan Chester	40	Engineer	1987-03-22	111

Restriction mode and joins

You can also create a restrictive result set by *joining* two or more tables. The rows of each table in the join can be used in the result set of a successive query. A table that is not named in a successive query is dropped from the result set. For more information on joins, see the *SQL Elements* chapter in the *SQL Language Reference*.

The SELECT command below builds a result set with rows from the tables T1 and T2.

SELECT A, B FROM T1, T2 WHERE ...;

This command takes the rows for table T2 in the current result set and joins them with the rows from the table T3.

SELECT C, D FROM T2, T3 WHERE ...;

## Limitations

You cannot use the following features while in restriction mode:

Aggregate functions	DISTINCT
GROUP BY	HAVING
UNION	ORDER BY
Stored commands	Stored procedures

## Bind variables

A bind variable, also called a program variable, refers to a data value associated with a SQL command. Bind variables associate (bind) a syntactic location in a SQL command with a data value that is to be used in that command.

Bind variables can be used wherever a data value is allowed:

- WHERE clause.
- VALUES clause in an INSERT command.
- SET clause of in UPDATE command.

Bind variables let you type a command once, replacing variable slots with the data that is to be executed with the command.

Bind variables in SQLTalk conform to the following guidelines:

- A bind variable name starts with a colon (:) and is followed by a number that refers to the relative position among the data items associated with the SQL command (such as :1, :2, :3).
- The data for each execution of a command can be on as many lines as is needed. SQLTalk reads an item of data for each bind variable in the SQL command.
- If there are more items of data on a line than there are bind variables, SQLTalk stops reading the line when it has read enough data.
- If the number of items on a line is less than the number of bind variables, SQLTalk continues reading on the next line until it has read enough data.
- Use commas to separate data items. You can indicate null values by two successive commas.
- SQLTalk strips leading and trailing blanks within a data field.

- Character strings containing white space do *not* need to be enclosed in double quote.
- Double quotes (") surrounding data are ignored if the double quote is the first character in the field. Otherwise, the double quote is interpreted as a character.
- The single quote is never a valid delimiter in a bind variable data string, and is *always* interpreted as a character.
- System keywords such as NULL, USER, SYSTIME, SYSDATE, SYSDATETIME cannot be used with inserts that utilize bind variables. However, they can be entered directly, as shown in this example:

```
INSERT INTO T1 VALUES (SYSDATETIME);
```

The following example illustrates the use of bind variables.

```
INSERT INTO EXPENSES (TYPE, WHAT, DATE, AMT)
VALUES (:1, :2, :3, :4)
\
HOUSE, TREE REMOVAL, 1/18/86, 469
BIZ, GRAPHICS CARD, 1/19/86, 110
MED, DENTIST, 1/20/86, 45
/
```

In this example, each variable in the VALUES clause is bound to an instance of data. For example, value :1 from the INSERT command is successively replaced with HOUSE, BIZ, and MED.

Similarly, each data value on a line corresponds with its equivalent bind variable. Each line of data is executed as an INSERT of one row.

Notice that the data replacing the bind variables :1, :2, :3, and :4 are entered in a group of lines. The backslash on a single line signals the start of data. The slash at the end of the data signals the completion of the input.

After the line with the backslash, SQLTalk displays the following message:

```
"PROCESSING DATA"
```

## Examples of bind variables

```
INSERT INTO TESTVARS (A_STRING, A_NUMBER)
VALUES (:1, :2)
\
"El Cid", 15000
El Cid, 15000
EL CID, ,
/
SELECT A_STRING, A_NUMBER FROM TESTVARS
```

```

A_STRING      A_NUMBER
-----
El Cid        15000
El Cid        15000
EL CID

UPDATE TABLE EXPENSES SET COMMENTS = :1 WHERE
    DATE = :2

\
"Beltran Tree Service", 1/1/94
"Hercules", 1/2/94
"Checkup", 1/3/94
/

```

## Entering long data with bind variables

Use bind variables to enter long data, such as LONG VARCHAR, binary, or bitmap. You can include the long data in the SQLTalk command or call it from a separate file.

When inserting long data into a column, precede it with the \$LONG keyword. You can then start entering data on the next line, and continue entering on successive lines. To mark the end of text, enter a double slash on a new line (//).

The following example illustrates inserting LONG VARCHAR data.

```

INSERT INTO BIO (NAME, BIO) VALUES (:1,:2)
\
SHAKESPEARE, $LONG
    William Shakespeare was born in Stratford-on-Avon on
    April 16, 1564. He was England's most famous poet and
    dramatist. . . . .
    He died in 1616, leaving his second best bed to his wife.
//

```

If the data for the LONG VARCHAR column comes from a file, enter the name of the file after the \$LONG keyword. In the following example, the text is found in various *.txt* files.

```

INSERT INTO BIO (NAME, BIO) VALUES (:1,:2)
\
SHAKESPEARE, $LONG shakes.txt
JONSON,$LONG jonson.txt
O'NEILL,$LONG oneill.txt
/

```

The next example shows how to insert multiple LONG VARCHAR column values within a row. This example creates a table and inserts data using a script file called *long.cmd*.

Following is the text of *long.cmd*:

```
CREATE TABLE X (A INT,B LONG VARCHAR,C LONG VARCHAR);
INSERT INTO X VALUES (:1,:2,:3)
\
1,$LONG,$LONG
one
//
ONE
//
2,$LONG,$LONG
two
//
TWO
//
/
```

Running *long.cmd* with the SQLTalk RUN command returns the following results:

```
RUN LONG.CMD;
CREATE TABLE X (A INT,B LONG VARCHAR,C LONG VARCHAR);
INSERT INTO X VALUES (:1,:2,:3)
1,$LONG,$LONG
2,$LONG,$LONG

SELECT * FROM X;
```

Entering RUN *long.cmd* performs the following:

```
      A
=====
      1
B
=
one
C
=
ONE
      2
B
=
two
C
=
TWO

2 ROWS SELECTED
```

# Error messages

This section describes the following information:

- The common message files called *error.sql* and *message.sql* that are shared by SQLBase client and server programs.
- The SQLBase error window.

## About error.sql

All SQLBase error messages are stored in a common error message file called *error.sql*. This file must be present on *all* client and server computers that run SQLBase software.

As the diagram below shows, each error message has message text, a reason, and a remedy.

```
00353 EXE NSY Object <name> specified in
      DROP SYNONYM is not a synonym
Reason: Attempting to execute a DROP
      SYNONYM and the named synonym is not
      a synonym but a table or view name.
Remedy: Modify the DROP SYNONYM
      statement to use a synonym name
      or if you really want to drop a
      table then use a DROP TABLE statement.
```

The error message text line contains an error code (in this case, 00353), a mnemonic (EXE NSY), and a message text (Not a synonym). When a program detects an error condition, it uses the error code to look up the error message.

## About message.sql

The message.sql file contains prompts, confirmations, and non-error messages. This file must be present on all client and server computers that run SQLBase software.

You can specify a directory in the SQLBASE environment variable where SQLBase can find message and error files. Otherwise, SQLBase uses this search order to find *message.sql* and *error.sql* (described in the previous section) on a client or server:

1. Current directory.
2. \SQLBASE directory on the current drive.

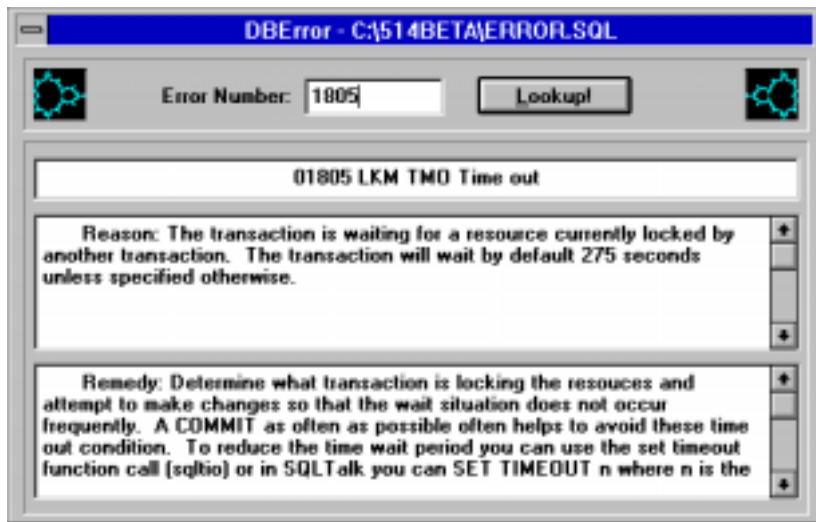
3. Root directory on the current drive.
4. Directories specified by the PATH environment variable.

## Displaying errors

SQLBase provides a window that displays the message text, reason, and remedy for a given error code. The program looks up this information in *error.sql*.

The error window program is installed on the client machine when you install SQLBase client software, and is assigned an icon in the client program group or folder.

To access the error window, click on the **DBError** icon.



*The DBError window*

To display information about a specific error, enter the error code in the Error Number field, and click **Lookup!**

## Setting the error level

You can control the level of detail in the error messages that SQLTalk displays by giving a SET ERRORLEVEL command. The error level that displays the least amount of detail (0) is the default.

## Chapter 2

# SQLTalk Command Reference

---

This chapter contains the syntax, description, and examples of each SQLTalk command. It is organized alphabetically by command name.

## SQLTalk command summary

Command	Function
ALTER COMMAND	Tells SQLBase to recompile a stored command the next time you EXECUTE it.
ALTER DBSECURITY	Changes database encryption key, database encryption level, and page alteration protection level
ALTER EXPORTKEY	Temporarily changes the key used to encrypt databases
BACKUP	Backs up database.
BEGIN CONNECTION	Establishes an explicit connection by issuing a connection handle to a specific database.
BREAK	Specifies breaks for a report.
BTITLE	Specifies a bottom title for a report.
COLUMN	Specifies column attributes for a report.
COMPUTE	Computes aggregates for a report.
CONNECT	Establishes an implicit connection to a database or establishes a cursor that is part of an implicit or explicit connection.
COPY	Copies a database or table.
DBERROR	Displays the text, reason, and remedy for a specified error code.
DISCONNECT	Disconnects an implicit connection to a database or disconnects a cursor that is part of an implicit or explicit connection.
EDIT	Edits command input with an external editor.
END CONNECTION	Terminates the specified connection which closes a connection handle.
ERASE	Erases a stored SQL command or procedure.
EXECUTE	Executes a stored SQL command or procedure, and recompiles if specified.
EXIT	Exits SQLTalk.
FETCH	Fetches rows from a result set.

Command	Function
LEFT	Begins the screen output to the left.
LIST	Displays an input command from edit buffer.
PAUSE	Pauses for a carriage return.
PERFORM	Executes a SQL command or procedure that has been prepared or retrieved.
PREPARE	Compiles (but does not execute) a SQL command.
PRINT	Prints a report to a printer or file.
RELEASE LOG	Releases the current log file.
REMARK	Displays a remark on the output screen.
REORGANIZE	Reorganizes the database.
RESTORE	Restores a backed-up database.
RETRIEVE	Retrieves a stored procedure/command for execution.
RIGHT	Begins the screen output to the right.
ROLLFORWARD	Recovers a database.
RUN	Runs a SQLTalk command file.
SAVE	Saves the report environment or a SQL command.
SAVE FILTER	Saves a set of restriction-mode ROW IDs.
SET	Sets options.
SET FILTER	Starts result set mode and restriction mode
SET SCROLL	Starts result set mode
SET SERVER	Establishes a server connection.
SET SPOOL	Records the SQLTalk session.
SHOW	Shows the value of a variable or attribute.
SHOW CONNECTION	Displays information for a specified connection.
SHOW DATABASES	Displays a list of databases.
SHUTDOWN	Shuts down the SQLBase server or a database.

Command	Function
SQLGET	Retrieves the values of many server and client parameters.
SQLSET	Sets the values of many server and client parameters.
STORE	Stores a precompiled SQL command.
TTITLE	Specifies the top title of a report.
UNDO	Reverts to the previous result set while in result set mode.
USE	Selects a cursor.

## ALTER COMMAND

➡➡ — **ALTER COMMAND** *command name* — **SET AUTORECOMPILE**

ON

OFF

 ⬅⬅

Use this command to flag a stored command with an AUTORECOMPILE setting. If you set this flag to ON and the stored command later becomes invalid, SQLBase automatically recompiles the stored command the next time you EXECUTE it.

A stored command becomes invalid when you alter its associated table or tables, such as dropping an index or column. Even if the table change does not directly affect the stored command, SQLBase still invalidates the command. You cannot use the stored command until you replace or recompile it.

You only set AUTORECOMPILE for stored commands created by users. SQLBase automatically recompiles all system-generated stored commands itself.

The UNLOAD and LOAD commands preserve the AUTORECOMPILE status for all stored commands.

To immediately recompile the stored command without having to wait until the next EXECUTE, execute the SQLBase RECOMPILE procedure, instead of running ALTER COMMAND.

## Clauses

### SET AUTORECOMPILE

If this setting is ON and the stored command later becomes invalid, SQLBase automatically recompiles the stored command the next time it is executed, either by the SQLTalk EXECUTE command or by the SQL/API *sqlret* function.

When you create a stored command, the default AUTORECOMPILE setting is ON.

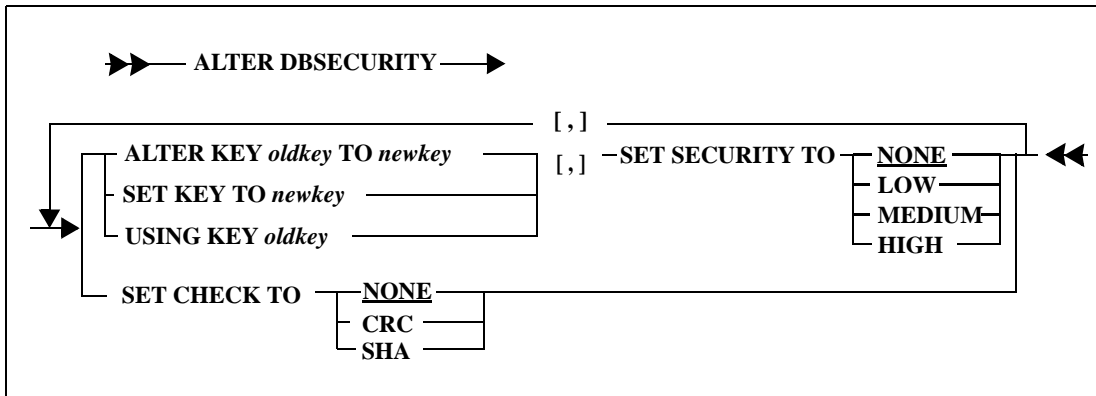
## Example

```
ALTER COMMAND ADDNAMES SET AUTORECOMPILE ON;
```

## See also

STORE  
EXECUTE

# ALTER DBSECURITY



This command changes the database encryption key, database encryption level, and page alteration protection level. You can checksum without setting the key. You can set security and set the key in any order, but if you set security, you *must* set the key.

**Note:** You must be logged on as SYADM and you must be connected to a database to perform this command.

---

**Warning:** Back up the database before executing this command. You can only perform this command when there are no other connected users.

---

This command changes the structure of the database. If a failure occurs during the command, you cannot recover the database. You cannot rollforward through this operation. Like a DROP DATABASE, this command deletes any transaction logs, regardless of the BACKUP LOGS setting.

---

**Warning:** You cannot rollforward past a change in the security configuration, including changing the server security password, the database page encryption, and database page alteration protection. If you try to do this, the restore stops just before the security change. After making a security change, you should restart your backup procedure by making a fresh backup of the database and its logs because previous versions of the database cannot be recovered beyond the security change. Therefore, do this after making a security change:

1. Shut down the server.
  2. Perform an offline backup
- OR

Start the server and perform an online backup.

---

To change the database key or the database encryption level, you must specify the old password in an ALTER KEY, SET KEY, or USING KEY clause.

Because some forms of this command can take a long time, SQLBase reports progress messages to the process activity window every 5,000 pages.

Decrypting and encrypting an entire database is a processor and disk I/O intensive operation. Performance of the server generally may be reduced during some forms of this operation.

---

**Important:** Since this is a SQL command, you can compile and execute this command through the SQL/API.

---

## Clauses

### SET KEY

Specify this clause to create a database encryption key for the first secure operation on the database.

---

**Important:** A new database does not have a database encryption key or a security encryption level and is compatible with non-secure versions of SQLBase. However, once you set a security

key for a database, it is not compatible with non-secure version of SQLBase, even if the security level is set to none.

---

To enable security features, give this command:

```
ALTER DBSECURITY SET KEY TO 'newkey'
```

to create a database encryption key.

After this command executes, the database is security-enabled and cannot be accessed by a non-secure version of SQLBase. However, until you choose an encryption method (via the SET SECURITY clause), the database is not secure.

Follow the same rules for creating a database key that you follow for long identifiers such as column names (18 character maximum).

## USING KEY

Specify this clause to set the key to the key used in the previous secure operation.

## ALTER KEY

Specify this clause to change the database encryption key.

## SET SECURITY TO NONE/LOW/MEDIUM/HIGH

Specify this clause to set the database page encryption level.

This process can take some time depending on the size of the database.

Because this operation can take some time, you should back up the database first. If there is some type of failure during the encrypting process (such as a power failure), you cannot recover the database.

The None level does not encrypt data.

The Low level converts each database page using a cryptogram (each character is replaced with a different character using a non-trivial formula). This level discourages the casual observer using a file viewer such as a hex editor but is not intended to be very secure. However, it offers reasonably high-performance.

The Medium level is very secure and has little impact on performance.

The High level is the most secure, with some impact on performance. Due to various governments' import and export regulations, this level of encryption may not be available in your version of SQLBase.

## SET CHECK TO NONE/CRC/SHA

Specify this clause to set the database page alteration protection level.

With the None level of protection, anyone can change a page in a database and SQLBase cannot detect it. If data page encryption is used, it would be difficult to determine exactly what to change the data to, but it still could be done.

With the 16-bit CRC (Cyclic Redundancy Check) level of protection, SQLBase can detect most changes to the database. The CRC is stored in the page before it is encrypted, preventing alteration of the CRC as well.

The SHA (Secure Hash Algorithm) level of protection provides a digital signature for each database page. SHA keys are impossible to reverse and alteration is easily detected. This level provides substantially more protection than CRC.

## Example

Set the database encryption key and page encryption level:

```
alter dbsecurity set key to key1 set security to high;
```

Set the database page alteration protection level:

```
alter dbsecurity set check to sha;
```

You can combine clauses in one command. For example, the commands in the previous three examples can be combined into one:

```
alter dbsecurity set key to key1, set security to high, set  
check to sha;
```

To turn off database page encryption for a database, enter a command like this:

```
alter dbsecurity using key key1 set security to none;
```

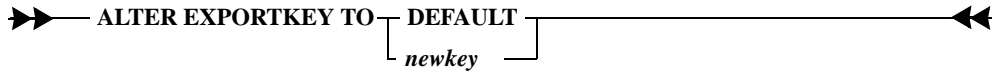
To turn off database page alteration protection, enter a command like this:

```
alter dbsecurity set check to none;
```

## See also

ALTER EXPORTKEY

# ALTER EXPORTKEY



```
ALTER EXPORTKEY TO { DEFAULT | newkey } [ ... ]
```

This command changes the server security password used to encrypt databases. You use this command to export (BACKUP) and import (RESTORE) databases to and from another server with a different server security password.

---

**Note:** You must be connected to a database to perform this command.

---

For BACKUP DATABASE, SQLBase encrypts the database file with the specified server security password instead of the default server security password. For RESTORE DATABASE, SQLBase decrypts the database file using the specified server security password instead of the default server security password.

During a restore, SQLBase tries to decrypt the database using the current server security password. If that password cannot decrypt the database, SQLBase uses the password specified by the most recent ALTER EXPORTKEY command. SQLBase then re-encrypts the database using the current server security password. During a backup, SQLBase uses the password specified by the most recent ALTER EXPORTKEY command to encrypt the database.

When you disconnect (SET SERVER OFF) or the server shuts down, the password reverts to the default.

---

**Important:** Since this is a SQL command, you can compile and execute this command through the SQL/API.

---

## Clauses

### DEFAULT

The normal server security password.

'newkey'

The temporary server security password.

## Example

To backup a database that will be restored on a different server with a different server security password, first give an ALTER EXPORTKEY command, specifying the other server's password:

```
alter exportkey to password2;
```

Then back up the database:

```
backup snapshot to d:\backup;
```

To restore a database from a server with a different server security password, first give an ALTER EXPORTKEY command, specifying the other server's password:

```
alter exportkey to password2;
```

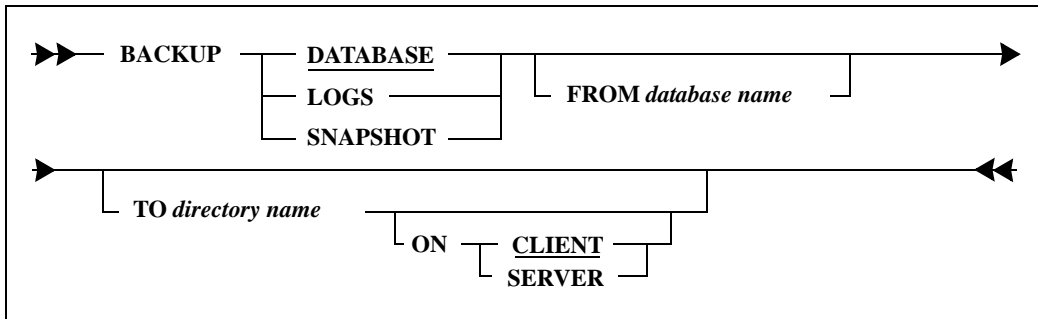
Then restore the database:

```
restore snapshot from d:\backup to island;
```

## See also

ALTER EXPORTKEY

## BACKUP



This command backs up a database, transaction log files, or both.

You can backup a database to a file with the following name:

```
database-name.bkp
```

Transactions that have been committed when the backup starts are included in the backup. Active transactions are not included.

For all options except SNAPSHOT, the following rules apply:

- The SET SERVER command must come before the BACKUP command.
- You must be logged in as DBA or SYSADM.

You do not need to be logged on as SYSADM to perform a BACKUP SNAPSHOT. This allows users other than SYSADM to backup the MAIN database.

Once you backup a database or log files to a directory, you can transfer the backup files to archival media and then delete them from the hard disk.

You can backup a partitioned database and restore it to a non-partitioned database, and vice-versa.

If you have a database greater than 2 gigabytes, you must perform backups to multiple segments. Each segment must be 2 gigabytes or less in size. To perform the backup operation in segments, you must specify a control file *databasename.BCF*. This control file describes the location and size of the segments to which you want to restore your database. Read the *Database Administrator's Guide* for details on the format of the control file.

Note that there is no need to explicitly provide the name of the file in the BACKUP command's syntax. If a control file (*databasename.BCF*) is present in the directory specified in the TO and FROM clause of the BACKUP command, SQLBase performs a segmented backup operation. If a control file is not present, SQLBase backs up the database to a single *databasename.BKP* file.

Note the following when you use the BACKUP command:

- SQLBase issues an error if you try to back up databases in segments larger than 2 gigabytes.
- If you do not include a segmented backup control file (*databasename.BCF*) in the directory that you specified in the TO and FROM clause of the command, an unsegmented backup file is created (if less than the 2 gigabytes limit).
- SQLBase issues an error if the control file you specify does not have a minimum aggregate size to account for the total database size.

Read the *Database Administrator's Guide* for details on setting the SIZE parameter for the control file.

- SQLBase does not verify the existence of disk space availability for any of the files you create.

Be sure that the directories you designate in the BACKUP command and in the control file have sufficient space. For backup commands with the snapshot option, also be sure that the directory specified in the command contains sufficient space for the backup of the log files involved in the operation.

To back up a database encrypted with a different server security password than the one currently set, you must first give an `ALTER EXPORTKEY` command. For more, read *ALTER EXPORTKEY* on page 2-9.

---

**Warning:** You cannot rollforward past a change in the security configuration, including changing the server security password, the database page encryption, and database page alteration protection. If you try to do this, the restore stops just before the security change. After making a security change, you should restart your backup procedure by making a fresh backup of the database and its logs because previous versions of the database cannot be recovered beyond the security change. Therefore, do this after making a security change:

1. Shut down the server.
2. Perform an offline backup

OR

Start the server and perform an online backup.

---

## Clauses

### DATABASE

Copies the database (*.dbs*) file from its current directory to the specified backup directory.

LOGBACKUP must be ON to perform BACKUP DATABASE. You should *never* back up a database without also backing up the log files with it.

### LOGS

Copies unpinned log files from the current log directory (by default, the database directory) to the specified backup directory. Once this completes successfully, SQLBase deletes the log files that were backed up from the current log directory.

You should back up log files from different databases to different directories since their file names could conflict.

LOGBACKUP must be ON to perform BACKUP LOGS.

### SNAPSHOT

Copies the database and the associated log files from their current directory to the specified backup directory. Since BACKUP SNAPSHOT backs up a single recoverable database, it is suggested that you perform this function into an empty directory.

BACKUP SNAPSHOT is the recommended way to backup a database and its log files. There is only one step (RESTORE SNAPSHOT) needed to bring a database and its log files to a consistent state.

Alternatively, the files produced by BACKUP SNAPSHOT can be restored and recovered with the individual commands RESTORE DATABASE, ROLLFORWARD, and RESTORE LOGS.

The SNAPSHOT option does not require the SYSADM logon and password. This means that other users besides SYSADM can backup the MAIN database by using BACKUP SNAPSHOT.

This command causes a log rollover which closes the current log file so that it can be backed up. This means that a RELEASE LOG command is not necessary.

If LOGBACKUP mode is turned on, the log file is pinned until it is backed up using BACKUP LOGS. The backup command unpins the log, not BACKUP SNAPSHOT.

You cannot perform a BACKUP SNAPSHOT while in Read-Only (RO) isolation level.

Note that BACKUP SNAPSHOT does not backup a single file; you must provide directory name, not a file name.

#### FROM database name

This clause specifies the name of the database.

If the FROM clause is omitted, the database is assumed to be the one currently connected to the active cursor.

#### TO directory name

This specifies the destination directory for the backed up files.

This backup directory pathname can refer to the client or the server computer. You can specify the pathname with the ON SERVER or ON CLIENT clauses.

If you omit the TO clause, the current directory on the client computer is used as the backup directory.

If the destination files of the backup already exist (and the PAUSE option is turned ON), you are prompted with the following message before the files are overwritten:

```
Backup file already exists.  
Overwrite it (Y/N)?
```

It is a good idea to place the backup on a different drive or device to isolate it from failure of the drive on which the database is located.

#### ON CLIENT or ON SERVER

This specifies whether the destination directory for the backed up files is on the client or the server. The default is ON CLIENT.

## Example

The following example assumes that you have already entered a SET SERVER command.

```
BACKUP DATABASE TO \DEMOBKUP;  
DATABASE BACKED UP  
RELEASE LOG;  
RELEASE LOG COMPLETED  
BACKUP LOGS TO \DEMOBKUP;  
2 LOGS BACKED UP
```

## See also

```
RELEASE LOG  
RESTORE  
SET SERVER
```

## BEGIN CONNECTION

```
➡➡ BEGIN CONNECTION connection_name databasename/username/password ⬅⬅
```

This command establishes a new connection to a specified database. When you supply the name of the database along with the username and password required for its access, this opens a connection handle which identifies the specified database. You can create cursors to perform specific SQL commands within the connection handle. A connection handle represents a single database connection and transaction.

If the databasename, username, and password are not specified, SQLTalk uses the databasename, username, and password of the current cursor connection.

The beginning of a new connection does not modify the current cursor. SQLTalk always maintains what it considers to be the current cursor, which is the cursor used for any SQL command execution. For details, read *Chapter 3, Connection Handles* of the *SQL/API Programming Language Reference*.

## Clauses

connection name

Specify the connection name, which can be no more than eight characters in length.

databasename/username/password

This specifies the database to connect to.

Regardless of how many database connections have been established, there is only one current database and one current cursor at any given moment. The current database and cursor is the one to which a connection was established at sign-on or by the last CONNECT or USE command. If the database name is not specified in a BEGIN CONNECTION command, it is assumed that the current database will remain unchanged.

If you are connecting to the MAIN database, specify the name of the server where the MAIN database is instead of the database name itself.

username/password

Each valid user of the database is assigned a username (authorization ID) and password.

The BEGIN CONNECTION command can establish only one connection to the database through a username.

If you do not specify a username/password, the connection takes place with the same username/password as the current cursor. The current cursor is the one that was established at sign-on or with the last CONNECT or USE command.

## Example

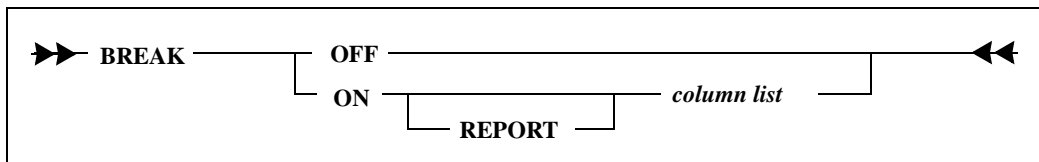
The following example creates a new connection:

```
BEGIN CONNECTION CH1 ISLAND/SYSADM/SYSADM;
```

## See also

END CONNECTION  
SHOW CONNECTION

# BREAK



This command suppresses repeating values for a specified column within a group. It inserts a blank line when the column changes value or after the last row of a report has been displayed.

If you entered the COMPUTE command, appropriate totals and subtotals are computed and printed for each group of rows separated by blank lines.

Once you enter the BREAK ON command, all subsequent SQL queries are displayed with the specified breaks. To add or remove a break specification, you must enter a new BREAK OFF command.

Unlike the COMPUTE and COLUMN commands, the BREAK command is not cumulative.

To turn off breaks, enter the BREAK OFF command. The BREAK OFF command also removes all specified COMPUTEs.

## Clauses

### ON/OFF

Turns BREAK capability on or off.

### REPORT

This specifies that a break occurs after the last row of the report. This is usually specified when an aggregate for the entire report needs to be computed (such as a grand total).

### column list

The column list contains column-IDs separated by one or more blanks.

Column-ids can be either:

- Position number of the column in a subsequent SELECT command.
- An alias specified with the COLUMN command.

The breaks occur in the same order as specified in the column list.

SQLTalk does not check to see if a SQL query results in a sorted order that is the same as the specified break order. You must ensure that a subsequent SELECT command contains an appropriate ORDER BY clause.

## Examples

The following command requests a break on the column DEPTNO.

```
BREAK ON 1;
SELECT DEPTNO, PROJNO, SUM(SAL) FROM EMP ORDER BY
      DEPTNO, PROJNO;
```

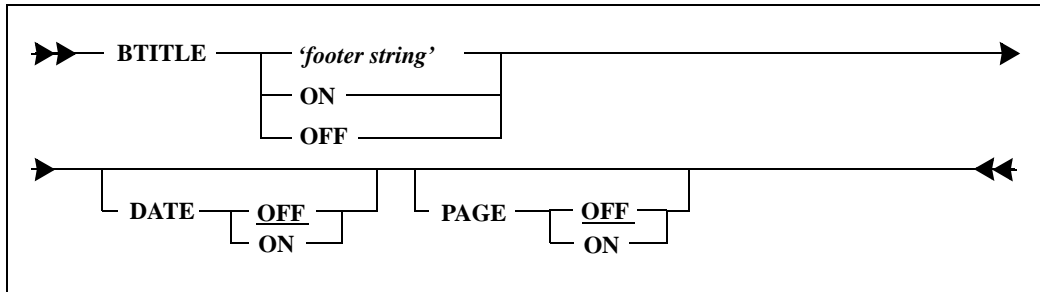
The following command requests a break on the columns DEPTNO and PROJNO, and on the last page of the report.

```
BREAK ON REPORT 1 2;
SELECT DEPTNO, PROJNO, SUM(SAL) FROM EMP ORDER BY
      DEPTNO, PROJNO;
```

## See also

COLUMN  
COMPUTE  
SHOW REPORT

## BTITLE



This command displays a title on the bottom of each page of a report. Once you have entered this command, the results of all subsequent SELECT commands are displayed with this bottom title.

You can change the bottom title by giving another BTITLE command or it can be turned off by giving the BTITLE OFF command. You can turn on BTITLE again with the BTITLE ON command.

This command is cumulative, meaning that once the title has been turned on, portions of it can be changed (such as adding a date or a page number).

## Clauses

'footer string'

You must enclose the footer string in *single* quotes.

The string can consist of up to three separate substrings, each delimited by the vertical bar ( | ) character. The | causes the substring that follows it to be displayed on a new line (multi-line footer).

All the lines of the footer are automatically centered within the display width of the page. The display width of the page is specified by the SET LINESIZE command.

### DATE

If this is ON, the current date appears on the bottom left hand corner of each page of the report. The date is displayed in *Mon dd, yyyy* format (such as April 25, 1996). The date is always displayed on the last line of the page, regardless of the number of lines in the bottom title.

### PAGE

If this is ON, the current page number is displayed on the bottom right hand corner of each page of the report as '*Page n*', where n is the page number. The page number is always displayed on the last line of the page, regardless of the number of lines in the bottom title.

## Example

Shown below is a BTITLE command and the resulting title.

```
BTITLE 'COMPANY CONFIDENTIAL|***' DATE ON
PAGE ON;
```

COMPANY CONFIDENTIAL		
April 23, 1996	***	Page 1

## See also

SET LINESIZE

SHOW REPORT  
TTITLE

## COLUMN

➡➡	COLUMN <i>column ID</i>	ALIAS	<u>OFF</u>	←←
			<i>alias name</i>	
		HEADING	<i>'column title'</i>	
		WIDTH	<u>OFF</u>	
			<i>data width</i>	
		PICTURE	<i>'picture string'</i>	
		NULLS	<i>'replacement string'</i>	
		ADJUST	<u>OFF</u>	
			LEFT	
			RIGHT	
			CENTER	
		DUP	<u>OFF</u>	
			ON	
		WRAP	<u>OFF</u>	
			ON	
		PRINT	<u>OFF</u>	
			ON	
		SUBTITLE	<u>OFF</u>	
			ON	
		NEWPAGE	<u>OFF</u>	
			ON	

This command sets attributes that control the display format and break processing for a column in a report.

After you have entered this command, the results of all subsequent SELECT commands reflect the attributes that you specified.

You can specify more than one COLUMN command for a single column. The effect of this is cumulative, except that a new COLUMN command overrides an attribute that you specified in a previous COLUMN command.

An empty string specified as a pair of *single* quotes (') removes a previously-defined attribute for a column.

## Clauses

### column ID

This specifies the column ID. Column IDs can be one of the following values:

- Position number of the column in a subsequent SELECT command.
- An alias specified with the COLUMN command.

### ALIAS OFF or ALIAS alias name

This specifies an alias name for a column. The alias is used as a column-ID.

The rules for naming an alias are the same as for SQL columns. An alias name must begin with a letter (A through Z and the special characters #, @, and \$) and must not exceed 18 characters. The alias name must *not* be enclosed in quotes.

### HEADING 'column title'

This specifies the heading for a given column.

The heading can consist of up to three separate substrings, each delimited by the vertical bar (|) character. The | is a delimiter that causes the following substring to be displayed on a new line.

The width of a column heading cannot be greater than the line size (specified by the SET LINESIZE command). If any of the substrings in the column title are wider than line size, the extra characters are truncated at display time.

The heading string must be enclosed in *single* quotes (').

To remove a previously-defined column heading and return to the default heading, define *column-title-string* to be two *single* quotes (').

### WIDTH OFF or WIDTH data width

Each column can have a width specified for the data. This is a number from 1 to line size.

If the data width is less than the specified or default width of the column heading, the display width of the column is the greater of the data width or the column-heading width.

If the column has a PICTURE specified for it, the data width is ignored.

If the data width is not specified, or if it is turned OFF, the display width is equal to the size of the data as stored in the database.

## PICTURE 'picture string'

This specifies the display format of the data in a column. The picture string is a character-string constant that describes the format that is used to display numeric and date columns.

SQLTalk supports free formatting in an output picture. Characters that represent numeric time and date elements are replaced on output with database values.

The PICTURE can be up to 40 characters in length.

Picture strings are enclosed in *single* quotes.

A picture specified for a character column is ignored.

### Numeric Pictures

A numeric picture string must represent a valid number. For example, commas must be spaced three to the left of the decimal point and only one decimal point is allowed per number.

If the number of significant digits in the input exceeds the number of significant digits in the picture string, the number is not displayed. Instead, the string '\*\*' is displayed, indicating an error.

If a number contains decimal digits but there are not decimal digits in the picture, the decimal digits are truncated.

The following table explains the picture characters that can be used for a numeric column.

Picture Character	Description
9	Displays a digit in this position.
.	Displays a decimal point in this position. A decimal point can appear only once in a picture.
,	Displays a comma in this position. Commas must be spaced three to the left of the decimal point.
Z	Replaces a leading 0 in this position with a blank (zero-suppression). This symbol must be to the left of any digit specification in the picture string.

Picture Character	Description
\$	Displays a dollar sign in this position. It can appear at the beginning of a picture or it can be used as a floating character with the \$symbol appearing adjacent to the most significant digit in a numeric field. The \$ symbol cannot appear to the right of a 9, Z, or decimal point in a picture-string.
-	Displays a minus sign if the value is negative. The minus sign is displayed to the left of the value. If the value is positive, a blank space is displayed to the left of the data.
E	Displays data in scientific notation. The default width for a scientific notation string is 10 characters.

Examples of Numeric Pictures

```
COLUMN AMT PICTURE '$ZZZZ.99';
```

Input	Output
.85	\$ .85
3.49	\$ 3.49

```
COLUMN AMT PICTURE '9999';
```

Input	Output
33.99	0034
1111	1111

Date/Time Pictures

The standard output of a date/time value in SQLTalk is illustrated in the following diagram.



You can substitute slashes or spaces for the hyphens in the diagram above.

For a DATE data type, the time portion is omitted and for a TIME data type, the date portion is omitted.

For example, January 12, 1996, 3:15 PM would be output as 12jan1996 03:15:00 PM.

Date and time format strings must be enclosed in *single* quotes.

Note that the time can be added to any of the date pictures to give date and time, or the time picture can be used alone.

The following table explains the picture characters that can be used for a date/time column.

Characters	Replaced By
MM	A 2-digit number that represents the month.
MON	A 3-character abbreviation for the month.
DD	A 2-digit number that represents the day of the month.
YY	The last two digits of the year.
YYYY	The four digits of the year.
HH	A two-digit number that represents hours in military time.
MI	A two-digit number that represents minutes.
SS	A two-digit number that represents seconds.
AM or PM	Two characters that represents AM or PM.
9 99 999 9999 99999 999999	A number with 1 to 6 digits that represents fractions of a second. Only the least-significant 6 digits are considered.

A backslash prevents substitution and forces the next character into the output from the picture. For example, the following picture:

Mo\mmy was born in YYYY

produces the output string of "Mommy was born in 1950" instead of "Mo04y was born in 1950."

If a date value is being formatted, time symbols such as HH or MI are not recognized. If a time value is being formatted, time symbols such as DD or MM are not recognized. All symbols are recognized for DATETIME (TIMESTAMP).

Examples of Date/Time Pictures

The date output for *July 14, 1996* is shown below with corresponding picture clauses.

COLUMN Command	Output
COLUMN 3 PICTURE 'dd/mm/yy'	14/07/96
COLUMN 3 PICTURE 'mm-dd-yy hh:mi:ss'	07-14-96 11:20:57
COLUMN 3 PICTURE 'Mo\mmys birthday is Mon dd'	Mommys birthday is Jul 14
COLUMN 3 PICTURE 'Mo\mmy\'s birthday is Mon dd'	Mommy's birthday is Jul 14

NULLS ‘replacement string’

This indicates that if the value of the column is null, it is displayed as a string of characters specified by replacement-string. Otherwise, the value is whatever is specified by the SET NULLS command (the default is blank).

The string must be enclosed in *single* quotes ('').

If the replacement-string is specified as two *single* quotes (''), this removes any previously-defined value for the replacement-string and returns it to the default.

ADJUST OFF/LEFT/RIGHT/CENTER

This indicates whether the data and column headings of a column are left-justified, right-justified or centered within the width of the column.

If the data is decimal, then centering means that the data is aligned on the decimal point and the decimal point is centered within the column width.

ADJUST OFF removes a previously-specified setting.

The default settings are:

Data Type	Adjustment
Character	LEFT
Integer	RIGHT
Decimal	CENTER

Data Type	Adjustment
Date/Time	CENTER

**DUP OFF/ON**

This attribute applies only to columns specified in the **BREAK** command. If this is **ON**, the data in this column is displayed for each row, regardless of value.

The default is **OFF**, which means that the data in the column is displayed only when its value changes (when a break occurs on the column).

**WRAP OFF/ON**

If this is **ON**, column data is wrapped for up to five successive lines. No attempt is made to wrap at word boundaries.

This option is only valid for character columns.

**PRINT OFF/ON**

If this is **OFF**, the column results are not displayed even if the query returned the results successfully.

All columns to the right of the column are adjusted leftward on the display screen.

Turning **PRINT ON** or **OFF** does not change the column-IDs of the displayed columns.

**SUBTITLE OFF/ON**

This attribute applies only to columns specified in the **BREAK** command. If this is **ON**, the data in the column specified by the **BREAK** is not displayed in rows and columns, but instead on a separate subtitle line when its value changes (when a break occurs on the column). The subtitle line includes the column heading on the left of the data.

This attribute can be **ON** for more than one column, in which case the column data is printed on the subtitle line in the order the columns are specified in the select list of the query.

The default is **OFF** which means that the data is not printed on a subtitle line.

**NEWPAGE OFF/ON**

This attribute applies only to columns specified in the **BREAK** command. If this is **ON**, the report skips to a new page after a break occurs for this column. If a **COMPUTE** command was given, the new page occurs after the computations have been displayed.

The default is OFF, which means that a break does not cause the report to start on a new page.

## Examples

These commands format the first column in the select list to be displayed with a heading of Employee Name with a width of 20 characters.

```
COLUMN 1 ALIAS EMPNAME;  
  
COLUMN EMPNAME HEADING  
      'EMPLOYEE |NAME' WIDTH 20;
```

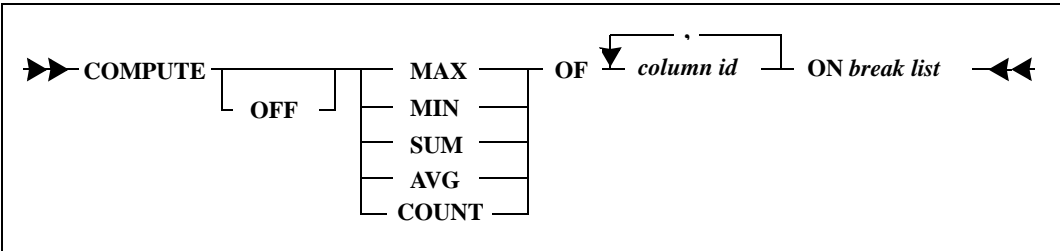
This command formats the second column in the select list.

```
COLUMN 2 HEADING 'BIRTHDATE'  
      PICTURE 'DD-MM-YY' ;
```

## See also

```
BREAK  
COMPUTE  
SET LINESIZE  
SET NULLS  
SHOW REPORT
```

# COMPUTE



This command performs aggregate computations at breakpoints and prints the results. The computations are:

- SUM (totals, subtotals)
- AVG
- MIN

- MAX
- COUNT

These computations should not be confused with the SQL functions that have similar names. The SQL functions calculate and display rows of *summary* information for groupings without displaying detail data. The COMPUTE functions calculate and display summary rows of data along with detail data.

Before running this command, you must first issue a BREAK command. If a COMPUTE is given without a BREAK command, an error message is displayed.

Each aggregate function requires a separate COMPUTE command.

COMPUTE commands are cumulative. This permits multiple calculations for a single column. For example, if a COMPUTE specifies MIN of a column, and a subsequent COMPUTE specifies MAX for the same column, then both the minimum and maximum for that column are calculated and printed.

## Clauses

### OFF

Specify OFF to turn off a specific COMPUTE option. To turn on the function again, the COMPUTE command must be given again for that function.

### MAX

This function calculates the maximum value of the items in the compute list for each group of rows specified in the break list.

The computed values are printed at each break point on one or more lines with a blank line separating the summary calculations from the detail rows.

### MIN

This function calculates the minimum value of the items in the compute list for each group of rows specified in the break list.

The computed values are printed at each break point on one or more lines with a blank line separating the summary calculations from the detail rows.

### SUM

This function calculates the total value of the items in the compute list for each group of rows specified in the break list.

The computed values are printed at each break point on one or more lines with a blank line separating the summary calculations from the detail rows.

## AVG

This function calculates the average value of the items in the compute list to be calculated for each group of rows specified in the break list.

The computed values are printed at each break point on one or more lines with a blank line separating the summary calculations from the detail rows.

Null values are not included in the calculation.

## COUNT

This function calculates the number of the items in the compute list for each group of rows specified in the break list.

The computed values are printed at each break point on one or more lines with a blank line separating the summary calculations from the detail rows.

Null values are not included in the count calculations.

## OF column ID

The column-ID can be:

- The position number of the column in a subsequent SELECT command.
- An alias specified with the COLUMN command.

The calculations are performed for each of the column-IDs in the compute-list.

## ON break list

This contains column IDs separated by blanks and/or the keyword REPORT. All items in the break list must also have been specified as break points in the BREAK command (but the reverse is not true). No computations are performed for break points that are specified in the COMPUTE command but not in the BREAK command. If the break order is different in the two commands, the order specified in the BREAK command takes precedence.

## Examples

The example below shows a sequence of commands that assigns aliases with the COLUMN command, sets BREAK fields, and finally specifies three calculations using the COMPUTE command.

```
CREATE TABLE X (DEPTNO INT, LNAME CHAR(10), PROJNO INT,  
                SAL INT);  
INSERT INTO X VALUES (1, 'Wayne', 200, 1000);  
COLUMN 1 ALIAS DEPTNO;  
COLUMN 2 ALIAS LNAME;  
COLUMN 3 ALIAS PROJNO;
```

```

COLUMN 4 ALIAS SAL;
BREAK ON REPORT DEPTNO PROJNO;
COMPUTE SUM OF SAL ON DEPTNO PROJNO REPORT;
COMPUTE AVG OF SAL ON DEPTNO;
COMPUTE COUNT OF LNAME ON PROJNO;

```

The following command requests a break on the columns DEPTNO and PROJNO in the **SELECT** command shown later.

```
BREAK ON REPORT 1 3;
```

Display salary totals by department (DEPTNO), project (PROJNO) and report.

```
COMPUTE SUM OF SAL ON 1 3 REPORT;
```

Display the average salary of each department. Give DEPTNO the alias DEPT.

```

COLUMN 1 ALIAS DEPT;
COMPUTE AVG OF SAL ON DEPT;

```

Display the number of people on each project.

```
COMPUTE COUNT OF LNAME ON PROJNO;
```

The following example uses the **BREAK** command and the **COMPUTE** command on the following query.

```
SELECT DEPTNO, LNAME, PROJNO, SAL FROM EMP;
```

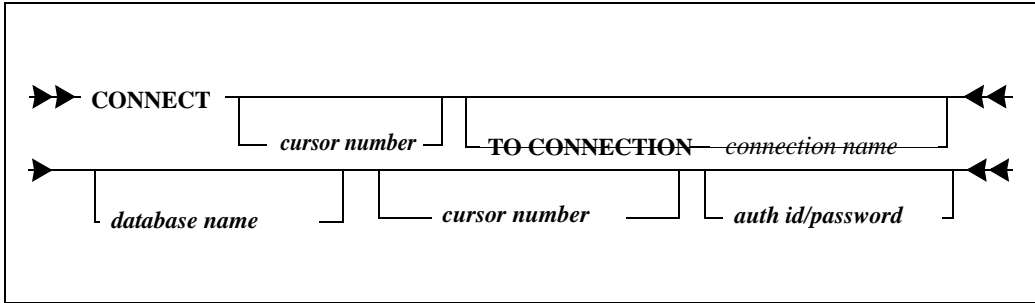
See also

```

BREAK
SHOW REPORT

```

# CONNECT



You can use this command to perform one of the following functions:

- Establish a cursor that is part of an implicit connection.  
When you connect a database to a specified cursor, SQLBase automatically assigns an *implicit connection handle* to the cursor. All cursors connecting to the same database, regardless of the username and authorization ID, are part of the same transaction.
- Establish a cursor that is part of an explicit connection.  
When you specify a cursor as part of an explicit connection, you must specify the connection name which identifies a connection handle. Each connection handle represents a specified database connection and a separate, independent transaction in the server. For details, read **BEGIN CONNECTION** on page 14.

Generally, cursors established implicitly apply to applications in which you are connecting cursors to a specific database that belong to a single transaction.

To create multiple, independent connections, SQLBase allows you to explicitly create multiple connection handles. You can use the **CONNECT** command to connect cursors to specific connection handles. Using explicit connection handles allow you to implement multiple transactions to the same database within an application. If you are using a connection handle and associating a cursor with the specified connection handle, you can only have one user per transaction.

---

**Note:** You can use both implicit and explicit connections within an application.

---

In SQLTalk, regardless of the number of connections established, there is only one current connection and one current cursor. The current cursor is the one that is executing the current SQL command and the current connection is the one to which the current cursor belong.

The current cursor and subsequently the current connection is the cursor established by the last **CONNECT** or **USE** command, or the cursor most recently established at SQLTalk start up time (if no cursor is specified at start up time, cursor 1 is the default). Cursors are associated with an implicit or explicit connection handle that identifies a connection to a specific database.

## Clauses

### database name

Specify a database name only if you are establishing an implicit database or cursor connection and want to specify a database to connect to.

If the database name is not specified in a **CONNECT** command, it is assumed that the current database will remain unchanged.

If you are connecting to the **MAIN** database, specify the name of the server where the **MAIN** database is instead of the database name itself.

### cursor number

A cursor number is required when you are establishing an explicit connection.

This is a unique number that identifies a single connection between SQLTalk and SQLBase. You can establish multiple cursors within an implicit or explicit database connection by giving each cursor in a connection a new cursor number.

If you are establishing an implicit connection to a database, you assign the cursor connection a specified database name. If you are connecting a cursor to an explicit database connection issued through a connection handle created with the **BEGIN CONNECTION** command, you assign the cursor connection a specified connection name. By connecting more than one cursor to the same database or connection, each cursor represents a SQL command or activity that is part of a single transaction.

If you use the **CONNECT** command to specify an existing cursor number, the old cursor is removed, and a new cursor is established with the specified parameters.

### connection name

Specify a valid connection name only if you are connecting a cursor to an explicit connection. An explicit connection is created with the **BEGIN CONNECTION** command, which issues a connection handle associated with a specific database connection. For details, read **BEGIN CONNECTION** on page 14.

### auth ID/password

If you are establishing an implicit connection to a database, you can assign each valid user of the database an authorization ID and password. For implicit

database connections, the `CONNECT` command can establish multiple connections to the database through multiple authorization IDs. This allows multiple users within one transaction.

Each authorization ID must be accompanied by its corresponding password (and separated by a `/`).

If you do not specify a authorization ID/password, the connection takes place with the same authorization ID/password as the current cursor. The current cursor is the one that was established at sign-on or with the last `CONNECT` or `USE` command.

## Examples

### Implicit connections

Connect a second cursor to the database `TESTDB` with John's authorization ID and password.

```
CONNECT TESTDB 2 JOHN/280Z;
```

Connect to a second cursor within the same database.

```
CONNECT 2;
```

### Explicit connections

Connect a second cursor to the connection named `CH2`.

```
CONNECT 2 TO CONNECTION CH2;
```

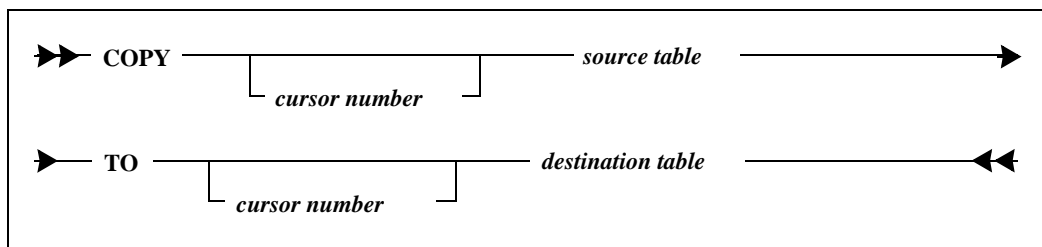
Connect to a third cursor within the same connection.

```
CONNECT 3 to CONNECTION CH2;
```

## See also

```
BEGIN CONNECTION  
GRANT (SQL command)  
SET CURSORNAME  
USE  
SHOW CONNECTION
```

# COPY



This command copies data from a source table to an existing destination table in the same or a different database.

COPY puts a shared lock on the source table, which means that other users can read the table, but are unable to modify it. The new data copied to the destination table gets an exclusive lock.

## Clauses

### cursor number

To copy data to a destination table on another database, prefix the destination and source table name with the cursor number that identifies a connection to the database.

If the cursor number is omitted, the system assumes that the source table and destination table exist on the current database. The current database is the one to which connection was established at sign-on or with the last CONNECT or USE command.

The default connection is with cursor 1.

### source table

The name of the table from which data is copied.

### destination table

The name of the table to which data is copied. *This table must exist.* The COPY command does not create the destination table. The columns of the destination table must be compatible in length and data type with the columns of the source table.

## Examples

Copy the employee data from the current database to the TEST database. The TEST database is connected by cursor 2.

```
COPY EMPLOYEES TO 2.EMPLOYEES;
```

Copy the employee data to another table, EMPLOYEE\_LIST, which is on the current database.

```
COPY EMPLOYEES TO EMPLOYEE_LIST;
```

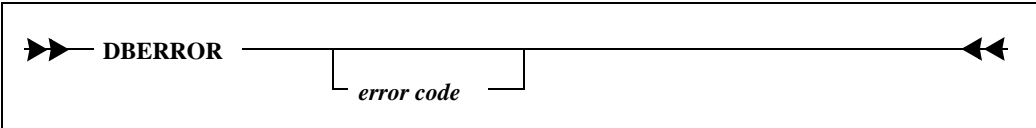
This example illustrates connecting to another database and copying from another table.

```
CONNECT PERSONEL 2 JENNY/BEAR;  
CREATE TABLE CODES (CODE CHAR(3),DESCRIPTION CHAR(50));  
COPY 1.CODES TO CODES;
```

## See also

```
CONNECT  
USE
```

# DBERROR



This command displays the message text, reason, and remedy code for an error code.

## Clauses

error code

To display information about a specific error code, enter the error’s associated numeric error code contained in the *error.sql* file.

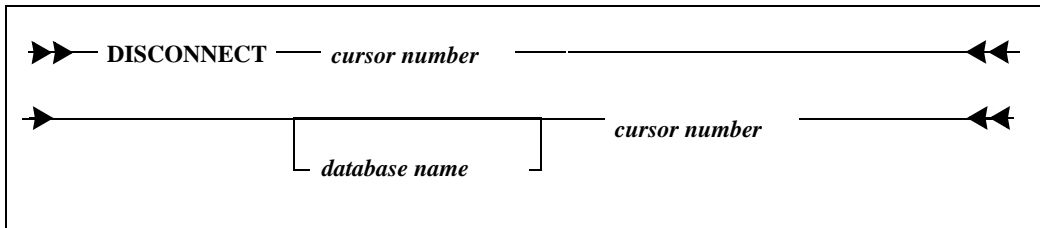
If you do not enter a specific error code, the command displays information about the last error code accessed.

## Example

The following example displays the message text, reason, and remedy for SQLBase error 1805.

```
DBERROR 1805;
01805 LKM TMO Time out
Reason: The transaction is waiting for a resource currently
locked by another transaction. The transaction will wait by
default 275 seconds unless specified otherwise.
Remedy: Determine what transaction is locking the resources and
attempt to make changes so that the wait situation does not
occur frequently. A COMMIT as often as possible often helps to
avoid these time out condition. To reduce the time wait period
you can use the set timeout function call (sqltio) or in SQLTalk
you can SET TIMEOUT n where n is the number of seconds. The
number of seconds can range from 1 to 1800 seconds (30 minutes).
```

## DISCONNECT



Use this command to disconnect a cursor that is part of an implicit or explicit connection handle by specifying the cursor number.

Note that the cursor that you specify cannot be the current cursor, which is the one you most recently connected to at SQLTalk start up time, or with the most recent CONNECT or USE command.

When you connect a database to a specified cursor, SQLBase automatically assigns an *implicit connection handle* to the cursor. When you use the DISCONNECT command to disconnect from this database, SQLBase automatically destroys the implicit connection handle to the cursor.

When closing the final cursor in an explicit connection handle, the transaction remains pending. Depending on the CLOSETRANSID option setting, it is either committed or rolled back when you terminate the connection handle using the END CONNECTION command. When closing the final cursor in an implicit connection handle, the transaction is either committed or rolled back depending on the

CLOSETRANSID option setting. For details on specifying connect closure behavior, read the CLOSETRANSID option described in the SET command in this chapter.

## Clauses

### cursor number

A cursor number is required whether you are disconnecting a cursor that was connected implicitly or explicitly.

Specify the cursor from which you are disconnecting. You cannot disconnect from the current cursor (the cursor that is currently being used).

### database name

If you are disconnecting an implicit connection, you can specify the name of the database for the specified cursor you are disconnecting. If you disconnect the database name, *all* cursors for the current database are disconnected. The current database is the database to which you are connected a sign-on or with the most recent CONNECT or USE command.

## Examples

Disconnect cursor 2. Note that this example can apply to either a cursor that was connected implicitly or explicitly.

```
DISCONNECT 2;
```

Disconnect DATABASE TESTDB that was implicitly connected to cursor 2.

```
DISCONNECT TESTDB 2;
```

## See also

```
BEGIN CONNECTION  
CONNECT  
END CONNECTION  
USE
```

# END CONNECTION

➡➡ **END CONNECTION** — *connection\_name* ————— ⬅⬅

This command terminates a specified connection that you created explicitly with the **BEGIN CONNECTION** command. You supply the name of the connection that you want to terminate. This closes the corresponding connection handle that is associated with the database, thereby terminating the connection.

Note that you cannot terminate the connection of the *current cursor*. SQLTalk always maintains what it considers to be the current cursor. The current cursor is the cursor used for any SQL command execution.

By default, the **END CONNECTION** command causes an implicit commit for a SQLBase database associated with the connection. Use the **SHOW CLOSETRANSMETHOD** to display whether a **COMMIT** or **ROLLBACK** will be issued before the connection handle is terminated. To change the setting to **ROLLBACK** for SQLBase databases, use the **SET CLOSETRANSMETHOD** command. For details, read the **CLOSETRANSMETHOD** option described in the **SET** command in this chapter.

---

**Note:** You can disconnect an implicit connection to a database or cursor using the **DISCONNECT** command. For details, read **DISCONNECT** on page 35.

---

## Clauses

connection name

Specify the connection name, which can be no more than eight characters in length.

## Example

The following example terminates a specified connection:

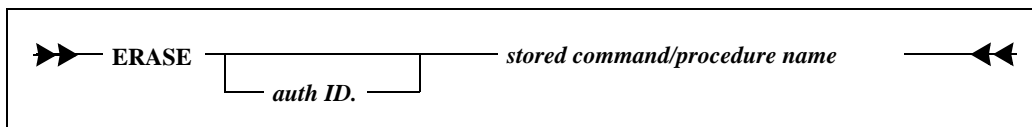
```
END CONNECTION CH2;
```

## See also

**BEGIN CONNECTION**

CONNECT  
USE  
SHOW CONNECTION

## ERASE



This command deletes a stored command or procedure.

### Clauses

*auth ID.*

If you are not the creator of the command or procedure, specify the authorization ID of the owner.

*stored command/procedure name*

The name of the previously compiled and stored command or procedure to delete.

### Examples

The following commands store a query and then erase it.

```
STORE myquery SELECT * FROM BIRTHDAYS;  
ERASE myquery;
```

The following command erases the *strdproc\_pres* stored procedure.

```
ERASE strdproc_pres;
```

### See also

EXECUTE  
STORE

# ERASE FILTER

➡➡ **ERASE FILTER** ————— *filter name* ————— ⬅⬅

This command erases a previously saved set of ROWID values in system tables, using a name that you previously specified in **SAVE FILTER**. Such ROWID sets are created when restriction mode is active.

More detail about restriction mode is available in Chapter 12, *Result Sets*, of the *SQLBase Advanced Topics Guide*.

## Clauses

*filter name*

This is a character string that must obey the constraints of a SQLBase long identifier.

## Examples

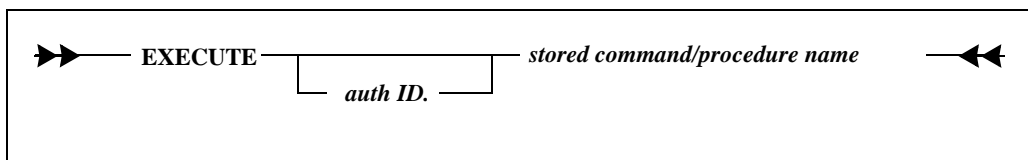
This script erases a set of restriction mode ROWIDs that had been saved earlier under the name 'Italia\_company'.

```
ERASE FILTER Italia_company;
```

## See also

**SET FILTER**  
**SAVE FILTER**

# EXECUTE



This command executes a stored command or procedure.

To manually recompile stored commands, use the RECOMPILE clause documented in Appendix B of the *Database Administrator's Guide*. This clause is itself a stored procedure provided by SQLBase. For more information on procedures, read the *Procedures and Triggers* chapter in the *SQL Language Reference*.

You cannot execute stored commands or procedures while in restriction mode. However, you *can* turn on result set and restriction modes *inside* of a procedure. In other words, you cannot save the results of a stored procedure in a result set, but you can manipulate result sets inside of a stored procedure.

Other users can execute your stored commands by retrieving them as *user.command name*. For example, if you stored a procedure as SYSADM, other users can retrieve it as *SYSADM.command*. The users must have access privileges to the tables involved.

If you are executing a static stored procedure, you must first store it with the STORE command. [SQLTalk User Guide](#) [Clauses](#)

auth ID.

If you are not the creator of the command or procedure, specify the authorization ID of the owner. For example, assume that SYSADM created a stored command. To execute it, enter this command:

```
EXECUTE SYSADM.command-name
```

stored command/procedure name

The name of the stored command or procedure to execute.

If the SQL command contains bind variables, the EXECUTE command must be followed by the input data. The delimiter for the EXECUTE command *must be* a line containing a backslash (\). This notifies SQLTalk that input data comes from the keyboard. The end of the input data is marked with a forward slash (/) on a new line.

The RECOMPILE procedure

SQLBase provides a procedure called RECOMPILE. Use this stored procedure to manually recompile a stored command. To run RECOMPILE for a specific

database, first run the *recomp.sql* script against the database. For information on this procedure, read Appendix B in the *Database Administrator's Guide*.

## Examples

This example compiles and stores a command ADDNAMES. The EXECUTE command contains bind variables, so you must enter data.

```
STORE ADDNAMES
  INSERT INTO FRIENDS (NAME) VALUES (:1);
EXECUTE ADDNAMES
\
PROCESSING DATA
LEN
MARGE
BETTY
/
3 Rows Inserted
```

The following commands store and execute the *pr\_pres* procedure.

```
STORE strdproc_pres
PROCEDURE pr_pres
Local Variables
  Sql Handle Curl
Actions
  Call SqlConnection (Curl)
  Call SqlStore (Curl, 'presname', \
'SELECT pres_name FROM sysadm.president')
  Call SqlDisconnect (Curl);

EXECUTE strdproc_pres;
```

The following example recompiles all of SYSADM's stored commands, both valid and invalid.

```
EXECUTE RECOMPILE
\
'SYSADM',%,_,0,0,0
/
```

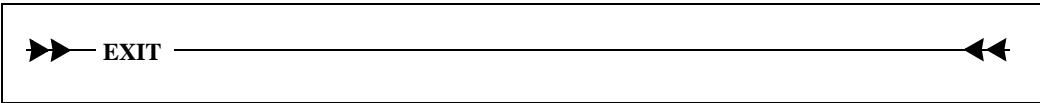
The following example shows possible output from this command:

RETURN_CODE	NUMBER_FOUND	NUMBER_DONE
-----	-----	-----
0	11	11

See also

ALTER COMMAND  
ERASE  
PREPARE  
STORE

# EXIT



This command exits SQLTalk and returns to the native operating system. It disconnects all cursors and causes an implicit COMMIT for all cursors and all databases.

## Example

```
EXIT;
```

# FETCH



**FETCH** *integer constant*



This command fetches the number of rows that you specified.

You can only use a FETCH in result set mode or after a PERFORM.

## Clauses

*integer constant*

This specifies the number of rows to retrieve, starting with the row pointed to by the last SET SCROLLROW command.

## Example

The example below uses an ADJUSTING clause and a result set.

```
SET CURSORNAME MYCUR;  
SET SCROLL ON;  
SELECT * FROM PRESIDENT;  
SET SCROLLROW 3;  
FETCH 10;
```

A different cursor is used to INSERT, preserving the result set.

```
CONNECT DEMO 2;
```

INSERT into result set.

```
INSERT INTO PRESIDENT (PRES_NAME)  
VALUES ('George Bush') ADJUSTING MYCUR;
```

Return to the result set cursor.

```
USE 1;
```



The result set is unaffected, so you can execute a FETCH without executing the SELECT again.

```
FETCH 5;
```

See also

PERFORM  
PREPARE  
SET SCROLL  
SET SCROLLROW

# LEFT

 **LEFT** *number of columns* 

This command shifts the displayed output of a SELECT command to begin with a column to the left of the current first column of the output.

This command enables you to view a wide report on a device whose LINESIZE is smaller than the DEVICESIZE.

This command does not affect the column-IDs of the columns in the select list.

## Clauses

**number of columns**

This indicates how many columns to the left of the currently displayed first column to shift the report.

If the specified number is greater than the number of columns existing to the left, then the report begins at the first column. Columns that you specified as PRINT OFF in the COLUMN command are ignored.

## Example

```
SELECT ITEMNO, ITEM, COST FROM INVENTORY;
```

ITEMNO	ITEM	COST
=====	=====	=====
1	CHOCOLATE	1.59
2	COCOA	2.48
3	COLA	1.46

First, the RIGHT command displays the rightmost column first.

```
RIGHT 2;
SELECT ITEMNO, ITEM, COST FROM INVENTORY;

COST
====
1.59
2.48
1.46
```

The LEFT command shifts the column, making the column to the left of the current first column the leftmost column.

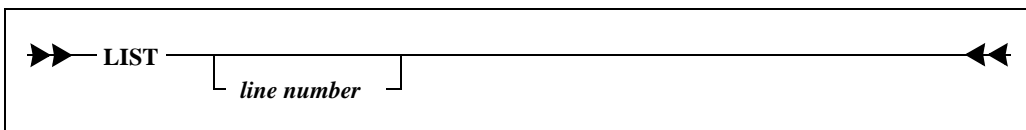
```
LEFT 1;
SELECT ITEMNO, ITEM, COST FROM INVENTORY;

ITEM      COST
=====
CHOCOLATE 1.59
COCOA     2.48
COLA      1.46
```

See also

COLUMN  
RIGHT  
SET DEVICESTR  
SET LINESTR

## LIST



This command displays the most recently-entered command. This command lets you view the input before editing it with the EDIT command.

## Clauses

line number

This indicates the line of the command to list. If you did not specify a line, the entire command is displayed.

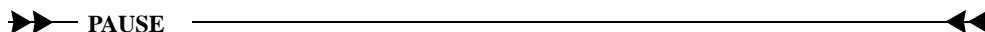
## Examples

```
SELECT A, B, C FROM TAB1
  WHERE A = 'N' ;
LIST;
SELECT A, B, C FROM TAB1
  WHERE A = 'N'
LIST 1;
SELECT A, B, C FROM TAB1
```

## See also

EDIT

## PAUSE



►► PAUSE ◀◀

This command suspends the SQLTalk session until you press the enter key.

This command can be used in a SQLTalk command file to cause a pause between groups of commands or to mark a transition point.

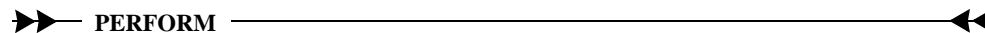
## Example

```
PAUSE ;
```

## See also

REMARK

## PERFORM



►► PERFORM ◀◀

This command runs a PREPARED (compiled) SQL command or procedure. You must execute the PERFORM command immediately after a PREPARE command.

You can bind data to variables before execution.

For SELECT commands, the PREPARE/PERFORM sequence only initializes for the first FETCH. SQLBase does not automatically fetch rows; you must explicitly request that rows be fetched with the FETCH command.

For SQL commands other than SELECT, the behavior of the PREPARE/PERFORM sequence is the same as if the SQL command had been entered from the keyboard or from a RUN file.

## Example

```
PREPARE PROCEDURE pr_pres
Local Variables
  Sql Handle Curl
Actions
  Call SqlConnect (Curl)
  Call SqlStore (Curl, 'presname', 'Select pres_name from \
    sysadm.president')
  Call SqlDisconnect (Cur 1);
PERFORM;
```

## See also

FETCH  
PREPARE  
SET SCROLL  
SET SCROLLROW

# PREPARE

➡➡ — **PREPARE** — *SQL command/procedure* ————— ⬅⬅

This command compiles a SQL command or procedure that is not stored, but does not execute it.

Four things happen when SQLBase compiles a SQL command or procedure:

1. The command itself or the SQL commands in the procedure are parsed. SQLBase detects syntax errors and verifies the existence of database objects.
2. SQLBase performs a security check.

3. SQLBase determines the execution plan, finding the optimal access path to the data.
4. SQLBase translates the command or commands into a series of executable modules.

If you do not precompile commands with PREPARE, you must compile them at run time. In a production application, precompilation makes complex SQL operations perform better.

## Clauses

### SQL command/procedure

Specify the command or procedure to compile. You cannot prepare a stored command or stored procedure.

## Example

The following example prepares a SQL command:

```
PREPARE SELECT * FROM CUST WHERE CUST_NAME = 'jones';
```

The following example prepares a procedure:

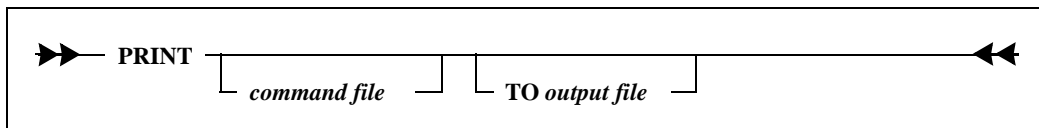
```
PREPARE
PROCEDURE Withdraw
  Parameters
    NUMBER:   nAccount
    NUMBER:   nAmount
    RECEIVE NUMBER:nNewBalance
  Local Variables
    STRING   sUpdate
    STRINGsSelect
  Actions
    SET sUpdate = 'UPDATE Checking SET \
      Balance = Balance - :nAmount ' || 'WHERE \
      AccountNum = :nAccount'
    CALL SQLImmediate(sUpdate)
    SET sSelect = 'SELECT Balance FROM Checking ' || \
      'WHERE AccountNum = :nAccount ' || \
      'INTO :nNewBalance'
    CALL SQLImmediate(sSelect);

PERFORM
\
100, 200, 0
/
```

## See also

EXECUTE  
 FETCH  
 PERFORM  
 SET SCROLL  
 SET SCROLLROW

## PRINT



This command prints a report on a printer or writes a report to an external file. When you enter the PRINT command, it causes the specified command file to be run (see the RUN command) and the results are printed or written to a specified file.

## Clauses

### command file

This file contains SQLTalk report formatting commands and one SELECT command. The file can be produced with the SAVE REPORT command. You can also produce a command file manually.

If any of the commands result in an error, an error message is displayed on the screen and the report is not produced.

All commands after the first SELECT are ignored.

If the command file name is omitted, the most recent SQL command is executed and the results printed.

### TO output file

This names the file where the report lines are written. If the TO clause is omitted, report lines are sent to the system printer.

## Examples

This command prints the results of the most recently-executed command on the system printer.

```
PRINT ;
```

This command prints the results of the command file MYREPORT on the system printer.

```
PRINT MYREPORT;
```



This command writes the results of the command file MYREPORT to the file MYREPORT.RPT.

```
PRINT MYREPORT TO MYREPORT.RPT;
```

See also

```
RUN  
SAVE REPORT
```

## RELEASE LOG

 **RELEASE LOG** 

A new log file is created automatically when the current active log file becomes full (this is referred to as log *rollover*). **RELEASE LOG** *forces* a log rollover and so is useful when executed just before a **BACKUP** command. In releasing the current active log file, you can back it up (if **LOGBACKUP** is enabled) and **SQLBase** can then delete it. In doing so, the most up-to-date backup is created.

You must issue a **SET SERVER** command before this command.

**RELEASE LOG** is not needed if the **BACKUP SNAPSHOT** command is used. **BACKUP SNAPSHOT** automatically forces a log rollover.

### Example

```
BACKUP DATABASE TO \DEMOBKUP;  
DATABASE BACKED UP  
RELEASE LOG;  
RELEASE LOG COMPLETED  
BACKUP LOGS TO \DEMOBKUP;  
2 LOGS BACKED UP
```

See also

BACKUP  
SET LOGBACKUP  
SET SERVER

## REMARK

➤➤ **REMARK** ————— ⬅⬅

This command is used in command files to display explanatory text. The text is displayed exactly as entered.

The REMARK command is echoed to the screen regardless of whether ECHO is ON or OFF.

### Example

```
REMARK  
\  
This is a comment. The comment starts with a backslash  
signalling data input. The comment ends with a forward slash  
and a new line.  
/
```

See also

PAUSE

## REORGANIZE

➤➤ **REORGANIZE** ————— ⬅⬅

This command reorganizes the current database using the following steps:

1. UNLOADs the database into a temporary file.
2. Deletes the database.
3. Initializes a new database.

4. LOADs the new database from the temporary file.

If recovery (logging) is enabled, REORGANIZE turns it off before reloading and back on after loading.

You should reorganize a database to correct file fragmentation. A database file can become fragmented from repeated changes and dropping of tables and so forth. In time, this fragmentation can affect the speed of database operations.

You must be the only user connected to the database. Other users cannot be connected to the database during a REORGANIZE.

A temporary file is used for unloading, called *sqltmp.nnn*. The *nnn* in the name is a unique serial number. The file is placed in the directory pointed to by the TMP environment variable. If the TMP environment variable is not defined, the file is placed in the current directory.

---

**Important:** 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.

---

A REORGANIZE operation retains all AUTORECOMPILE settings.

If you have changed the SYSADM password, a subsequent REORGANIZE command retains these new settings.

Unlike a LOAD or UNLOAD command, the unload/load processing for a REORGANIZE command can only take place on the client, not the server.

IMPORTANT: REORGANIZE is a SQLTalk construct that combines a series of operations in their simplest format. For some databases and situations it is necessary to use the individual component statements used by REORGANIZE to gain more control. Typically these situations include:

- Larger databases exceeding 1 gigabyte, where the unload file must be segmented into smaller files to avoid creating an individual file greater than 2 gigabytes and/or to use multiple drives.
- The extra safety of backing up the database unload file(s) is required (recommended).

---

**Warning:** Do not attempt an UNLOAD (explicitly or implicitly using REORGANIZE) that creates an unload file that exceeds 2 gigabytes. A LOAD operation using such a file is very likely to fail. REORGANIZE is not a safe operation if there is any likelihood that the unload file it generates will exceed 2 gigabytes.

---

## REORGANIZE with encrypted databases

When you perform a REORGANIZE, SQLBase does an unload (unencrypted) and then creates a new *unencrypted* database. Gupta Software recommends that instead of performing a REORGANIZE with encrypted databases, you manually reorganize by following the steps below:

1. Perform a UNLOAD command.
2. DROP the database.
3. Create a new database.
4. Encrypt the new empty database
5. LOAD the database.
6. Delete the UNLOAD file

---

**Warning:** Unload files are unencrypted and if you intend to retain them you should take steps to protect them such as by using the encryption feature of a file compression utility.

---

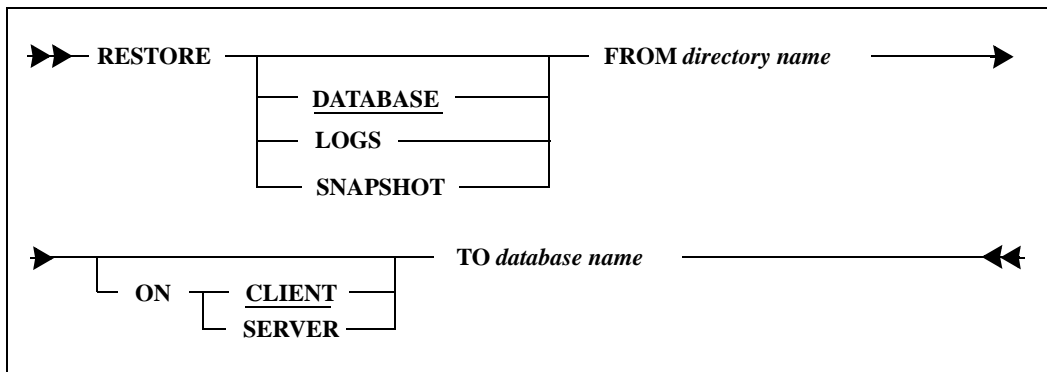
### Example

```
REORGANIZE ;
```

### See also

```
CHECK DATABASE
```

## RESTORE



This command restores a database, one or more transaction logs, or both. The database is always restored from the following file:

*database-name*.BKP

A RESTORE cannot be performed while users are connected to the database (including the DBA doing the RESTORE). To handle this, execute a SET SERVER command to switch your connection to the server rather than the database.

Note if you are using batch mode, you need to first specify NOCONNECT in the SQLTalk command line to prevent connecting to the database:

```
C:\Gupta> SQLTALK BAT NOCONNECT
```

Then after SQLTalk starts, you enter a SET SERVER command to connect to the server.

You can restore and recover backups made with BACKUP SNAPSHOT in one of two ways:

- A RESTORE SNAPSHOT command (the easiest way).
- A RESTORE DATABASE command followed by a subsequent ROLLFORWARD command.

You can restore a non-partitioned database to a partitioned database, and vice versa.

If you have a database greater than 2 gigabytes, you must perform the restore in multiple segments. Each segment must be 2 gigabytes or less in size. To perform the restore operation from segmented backups, you must specify a control file *dbname.BCF* (created from a successful segmented database backup). This control file describes the location and size of the segments to which you want to restore your database. Read the *Database Administrator's Guide* for details on the format of the control file.

Note that there is no need to explicitly provide the name of the file in the RESTORE command's syntax. If the restore control file is present in the directory specified in the TO and FROM clause of the RESTORE command, the restore operation is performed from the backup segments specified in the control file. Otherwise, the RESTORE command expects to find the restore file *dbname.BKP* to restore the database.

Note that SQLBase issues an error if you try to restore databases in segments larger than 2 gigabytes.

To restore a database encrypted with a different server security password than the one currently set, you must first give an ALTER EXPORTKEY command. For more, read *ALTER EXPORTKEY* on page 2-9.

---

**Warning:** You cannot rollforward past a change in the security configuration, including changing the server security password, the database page encryption, and database page alteration protection. If you try to do this, the restore stops just before the security change. After making a security change, you should restart your backup procedure by making a fresh backup of the database and its logs because previous versions of the database cannot be recovered beyond the security change. Therefore, do this after making a security change:

1. Shut down the server.
  2. Perform an offline backup
- OR

Start the server and perform an online backup.

---

## Clauses

### DATABASE

Copies the database backup to the current database directory.

### LOGS

Copies the transaction log files associated with a database to the current log directory and applies them to the restored database. This command continues restoring logs until all the logs in the backup directory that need to be applied have been exhausted.

After each execution of **RESTORE LOGS**, a message showing the next log file to be restored is displayed. If there are more logs to be processed than can fit on disk at one time, you can use the **RESTORE LOGS** command repeatedly to process all the necessary logs.

If a log file with the same name already exists in the current directory, you are prompted with this message:

```
Log file already exists. Overwrite it (Y/N)?
```

If the log file requested is not available, you can use the **ROLLFORWARD database** **END** command to terminate media recovery and recover the database using the information obtained up to that point (if possible).

### SNAPSHOT

This option restores the database and the associated log files that were created with a **BACKUP SNAPSHOT** command. The log files, if there are any, are automatically applied to the database. Therefore, a subsequent **ROLLFORWARD** command should never follow a **RESTORE SNAPSHOT**.

### FROM directory name

This specifies the directory name where the files to be restored are located.

### ON CLIENT or ON SERVER

This specifies the directory name where the backup files to be restored are located. This directory can be on the client or server. The default is ON CLIENT.

### TO database

This specifies the name of the database that is to be restored.

If a database file with the same name already exists, you are prompted with this message:

```
Database file already exists. Overwrite it (Y/N)?
```

## Example

```
RESTORE DATABASE FROM \DEMOBKUP TO DEMO;  
DATABASE RESTORED
```

```
ROLLFORWARD DEMO;  
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 \DEMOBKUP TO DEMO;  
4 LOGS RESTORED
```

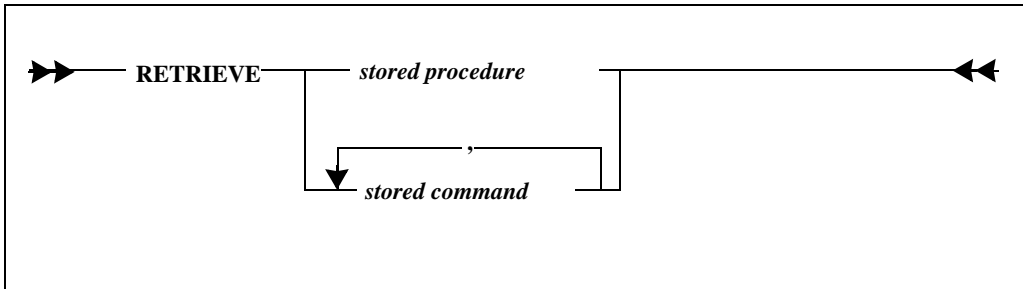
The log file 5.LOG could not be found. Use the RESTORE LOGS command to restore this log and continue the rollforward process. If this log is not available, use the ROLLFORWARD database END command to complete the recovery process.

```
ROLLFORWARD DEMO END;  
ROLLFORWARD COMPLETED
```

## See also

BACKUP  
RELEASE LOG  
ROLLFORWARD  
SET SERVER

# RETRIEVE



This command retrieves a stored SQL command or stored procedure, but does not execute it. Once a command or procedure has been retrieved, data can be bound if needed and the command/procedure can be executed.

Once a command or procedure is retrieved, it cannot be destroyed by a commit or rollback.

If another transaction changes the system catalog items that the retrieved command or procedure depends on between the commit and the execute, the execute fails.

You cannot use stored commands/procedures while in restriction mode.

## Clauses

stored command/procedure

Specify the command or procedure to retrieve. You can only retrieve one stored procedure at a time. You can retrieve multiple stored commands, but the last command in the series must be a **SELECT** command.

## Example

The following example stores, retrieves, and performs a SQL command. Since the stored command contains a **SELECT** statement, you must issue a **FETCH** command after the **PERFORM** to retrieve the output rows.

```

STORE sceng SELECT * FROM ENGINEERS;
RETRIEVE sceng;
PERFORM;
FETCH 10;
EMPL_NUM  NAME                REP_OFFICE  TITLE      HIRE_DATE
=====  =====

```

100	Paul Atkins	10	Manager	12-FEB-1988
10	Bob Smith	20	Sen.Engineer	05-SEP-1992
107	Murray Roch	30	Sen.Engineer	25-JAN-1991
102	Larry Sanchez	10	Sen.Engineer	12-JUN-1989
101	Sheila Brown	10	Engineer	10-OCT-1990
06	Sam Valdez	30	Manager	20-APR-1990
105	Rob Jones	20	Engineer	08-SEP-1991
103	Anna Rice	20	Manager	10-JUL-1985
108	Mary Adams	40	Manager	10-AUG-1988
109	Nancy Bonet	40	Sen.Engineer	12-NOV-1989

10 ROWS SELECTED

The following example stores, retrieves, and performs a stored procedure:

```
STORE procwd
  PROCEDURE WithDraw
  Parameters
  NUMBER:  nAccount
  NUMBER:  nAmount
  RECEIVE NUMBER:nNewBalance
  Local Variables
  STRING   sUpdate
  STRING   sSelect
  Actions
  SET sUpdate = 'UPDATE Checking SET \
    Balance = Balance - :nAmount ' || 'WHERE \
    AccountNum = :nAccount'
  CALL SQLImmediate(sUpdate)
  SET sSelect = 'SELECT Balance FROM Checking ' || \
    'WHERE AccountNum = :nAccount ' || \
    'INTO :nNewBalance'
  CALL SQLImmediate(sSelect);
  RETRIEVE procwd;
  PERFORM
  \
  100, 200, 0
  /
```

See also

```
EXECUTE
FETCH
PERFORM
PREPARE
SET SCROLL
SET SCROLLROW
```

# RIGHT

➡➡ **RIGHT** *number of columns* ⬅⬅

This command shifts the displayed output of a SELECT command to the right of the first column.

This command enables you to view a wide report on a device whose LINESIZE is smaller than the DEVICESTR.

This command does not affect the column-IDs of the columns in the select list.

## Clauses

### number of columns

This indicates how many columns to the right of the currently displayed first column the report must begin. If the specified number is greater than the number of columns existing to the right, no columns are displayed. Columns that are specified as PRINT OFF (see COLUMN command) are ignored for purposes of this command.

## Example

```
SELECT ITEMNO, ITEM, COST FROM INVENTORY;

ITEMNO      ITEM      COST
=====
1           CHOCOLATE  1.59
2           COCOA    2.48
3           COLA     1.46
```

First, the RIGHT command displays the rightmost column first.

```
RIGHT 2;
SELECT ITEMNO, ITEM, COST FROM INVENTORY;

COST
====
1.59
2.48
1.46
```

The LEFT command shifts the column, making the column to the left of the current first column the leftmost column.

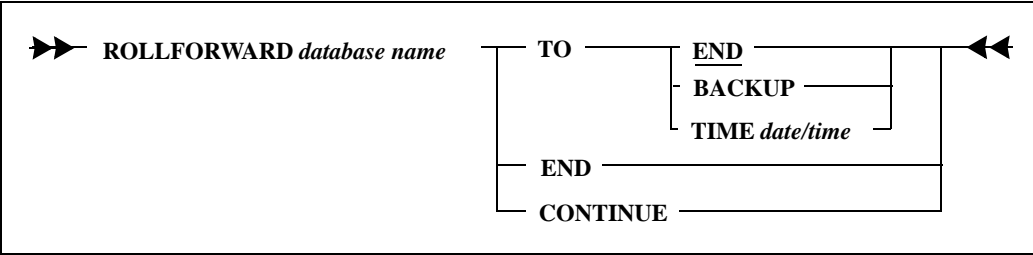
```
LEFT 1;
SELECT ITEMNO, ITEM, COST FROM INVENTORY;

ITEM      COST
=====
CHOCOLATE 1.59
COCOA     2.48
COLA      1.46
```

## See also

```
COLUMN
LEFT
SET DEVICESTR
SET LINESTR
```

# ROLLFORWARD



This command brings a database backup to a consistent and up-to-date state by applying log files to it after a RESTORE.

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*. The *4.log* and *5.log* log files cannot be applied to the database. An unbroken sequence of log files is required by recover a database backup to its most consistent state.

The SET SERVER command must precede a ROLLFORWARD command.

---

**Warning:** You cannot rollforward past a change in the security configuration, including changing the server security password, the database page encryption, and database page alteration protection. If you try to do this, the restore stops just before the security change. After making a security change, you should restart your backup procedure by making a fresh backup of the database and its logs because previous versions of the database cannot be recovered beyond the security change. Therefore, do this after making a security change:

1. Shut down the server.
2. Perform an offline backup

OR

Start the server and perform an online backup.

---

## Clauses

### TO END

This option rolls forward through all log files available. This option is the default because users normally want to recover as much of their work as possible.

## TO BACKUP

This option rolls forward to the end of the backup restored. This recovers all committed work up to the point of the last database backup.

## TO TIME

This option rolls forward to a specified date and time. The date and time are specified in the format "mm/dd/yy hh:mi:ss".

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.

## END

If a log file requested is not available, you can enter a `ROLLFORWARD database END` command to complete recovery with only the information obtained up to that point.

## CONTINUE

Use this option when restoring logs manually with a mechanism other than `RESTORE LOGS` (such as with a tape drive that is directly attached to the server that is used for restoring logs).

You should also use this option when the rollforward operation stops because SQLBase cannot find a log file that it needs. Use the `RESTORE LOGS` command (or *sqlrlf* SQL/API function) to copy the log files needed from the backup directory to the current log directory. Then enter a `ROLLFORWARD database CONTINUE` command (or *sqlcrf* SQL/API function) to resume the rollforward operation.

## Example

```
RESTORE FROM \DEMOBKUP TO DEMO;  
DATABASE RESTORED  
ROLLFORWARD DEMO;  
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 \DEMOBKUP TO DEMO;  
4 LOGS RESTORED
```

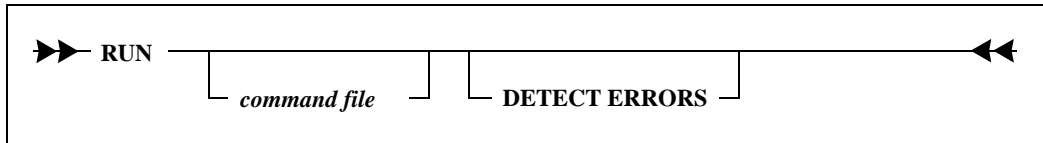
The log file 5.LOG could not be found. Use the RESTORE LOGS command to restore this log and continue the rollforward process. If this log is not available, use the ROLLFORWARD database END command to complete the recovery process.

```
ROLLFORWARD DEMO END;
ROLLFORWARD COMPLETED
```

## See also

BACKUP  
RESTORE  
SET SERVER

## RUN



This command executes the commands stored in the specified file.

## Clauses

### command file

This is the name of the command file that is executed. If the command file name is omitted, the RUN command executes the most recent SQL command.

If the command file contains SQLTalk report formatting commands, the environment created by the command file remains for successive SQL commands entered at the keyboard until changed by other environment commands.

The backslash (\) character marks the start of input data. The end of the data is marked with a forward slash (/) character on a new line followed by a carriage return.

The file can contain nested RUN commands. RUN commands can be nested within command files up to ten levels deep.

If the command file contains SQL commands that use bind variables, the input data can come from:

- The command file

- The keyboard
- A separate data file

### \$DATA Keyword

If the data is not included in the command file, the keyword \$DATA can be used to signal input from another source. This source is always a previous file, which can be a text file or the keyboard. When this keyword is encountered, SQLTalk looks for a backslash, followed by data lines, in the previous file.

### DETECT ERRORS

An error during command file processing causes execution to be interrupted. Otherwise, the error message is displayed and the next command in the command file is executed.

## Examples

### Data and commands in one file

The following command file (named *myproc1.cmd*) contains a SQL command and data:

```
UPDATE ITEMS SET PRICE =:1 WHERE ITEM_NAME = :2
\
.89, rice
.85, spaghetti
.83, rigatoni
/
```

This command executes the *myproc1.cmd* command file:

```
RUN MYPROC1.CMD;

PROCESSING DATA

.89, rice
.85, spaghetti
.83, rigatoni

3 ROWS UPDATED
```

### Take Input from the Keyboard

The following command file (named *myproc2.cmd*) contains a SQL command. The \$DATA keyword indicates that you will enter data at the keyboard:

```
UPDATE ITEMS SET PRICE =:1 WHERE ITEM_NAME = :2
\
$DATA
/
```

The following session updates the prices for items using the above command file. First, execute the RUN command and terminate it with the standard delimiter.

```
RUN MYPROC2.CMD ;
```

The SQL command is displayed:

```
UPDATE ITEMS SET PRICE = :1 WHERE ITEM_NAME = :2
```

At this point, SQLTalk has read the \$DATA keyword variable from the command file and is looking for a backslash from the keyboard to signal the beginning of input data. The backslash and data are entered. You can enter a forward slash to signal the end of input data:

```
\
1.50,tortellini
.89,macaroni
.99,linguine
/
3 ROWS UPDATED
```

## Take Input from Another File

The command file *myproc3.cmd* uses the data in *myproc3.dat*. The *myproc3.dat* file contains the data, so that when the \$DATA keyword is encountered in *myproc3.cmd*, it goes back to the previous source (*myproc3.dat*) for data.

The contents of *myproc3.dat* are:

```
RUN MYPROC3.CMD
/
\
1.85,tortellini
.69,macaroni
.85,linguine
/
```

The contents of *myproc3.cmd* are:

```
UPDATE ITEMS SET PRICE = :1 WHERE ITEM_NAME = :2
\
$DATA
/
```

RUN *myproc3.dat* in the SQLTalk window:

```
RUN MYPROC3.DAT ;
RUN MYPROC3.CMD
UPDATE ITEMS SET PRICE = :1 WHERE ITEM_NAME = :2

PROCESSING DATA
```

```
1.85,tortellini
.69,macaroni
.85,linguine
3 ROWS UPDATED
```

## Files with Multiple SQL Commands

In this example, two SQL commands require data input from another file. The data is in *myproc4.dat* and the SQL commands are in *myproc4.cmd*.

The file *myproc4.dat* contains the following lines:

```
RUN MYPROC4.CMD
/
\
1.00,bread
.75,tuna
.97,cheese
/
\
2.25,tortellini
/
```

The file *myproc4.cmd* contains the following lines:

```
INSERT INTO ITEMS (PRICE, ITEM_NAME) VALUES (:1, :2)
\
$DATA
/
UPDATE ITEMS SET PRICE =:1 WHERE ITEM_NAME =:2
\
$DATA
/
```

Run *myproc4.dat* in the SQLTalk Window:

```
RUN MYPROC4.DAT;
RUN MYPROC4.CMD
INSERT INTO ITEMS (PRICE, ITEM_NAME) VALUES (:1, :2)

PROCESSING DATA
1.00,bread
.75,tuna
.97,cheese
3 ROWS INSERTED

UPDATE ITEMS SET PRICE =:1 WHERE ITEM_NAME =:2
```

```

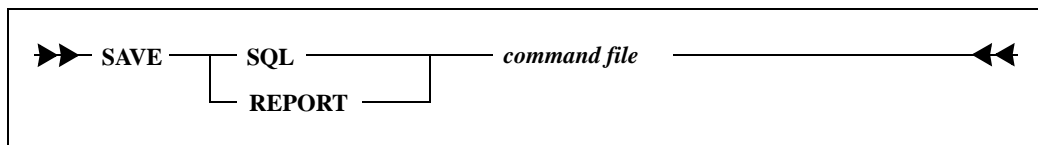
PROCESSING DATA
2.25,tortellini
1 ROW UPDATED

```

See also

SAVE  
SET PAUSE

## SAVE



This command writes the most recently executed SQL command or a series of SQLTalk commands to a command that can then be RUN. You can save the current SQL command or the current SQLTalk environment.

## Clauses

### SQL

If you specified this, the most recent SQL command is saved in the specified file.

### REPORT

If you specified this, all the SQLTalk commands making up the current display environment and report format are saved in the specified file. These include SET, BTITLE, BREAK, COLUMN, COMPUTE, and TTITLE commands. Not all SET options are saved (such as TIME, CURSOR, or PAUSE) since they do not affect report format.

You can save the default SQLTalk environment by entering the SAVE REPORT command during a session before making any formatting changes.

### command file

This is the name of the command file in which to store the commands. If the file already exists, you are prompted to append to the file, to abort the command, or to continue (write over the file). You can use the append feature to add a SQL command to an existing report environment file.

## Examples

This command changes the SQLTalk environment and writes the report commands to a file.

```
SET LINESIZE 60;
COLUMN 1 HEADING 'New Customers';
SAVE REPORT NEWCUST.TLK;
```

The following SQL command is executed and saved in the file NEWCUST.SQL.

```
SELECT CUSTNAME FROM CUSTOMER WHERE
    DATE >@MONTH(SYSDATE) - 1;
SAVE SQL NEWCUST.SQL;
```

To include the above command in the report NEWCUST.TLK, execute the following command:

```
SAVE SQL NEWCUST.TLK;
```

The following prompt is displayed. If you type 'P', the above SQL command is appended to NEWCUST.TLK.

```
File NEWCUST.TLK already exists. Type 'P' to append, 'C' to
continue....
```

## See also

```
BREAK
BTITLE
COLUMN
COMPUTE
RUN
SET
TTITLE
```

## SAVE FILTER

**➡➡ SAVE FILTER** ————— *filter name* ————— **◀◀**

This command writes the most recently retrieved set of ROWID values into system tables, using a name that you specify. You can retrieve these ROWID values later by using that name with the SET FILTER command. In addition, this command

performs the same functions as SET RESTRICTION OFF and SET SCROLL OFF, ending both restriction mode and result set mode.

This command is only valid when restriction mode is active. More detail about restriction mode is available in Chapter 12, *Result Sets*, of the *SQLBase Advanced Topics Guide*.

## Clauses

filter name

This is a character string that must obey the constraints of a SQLBase long identifier.

## Examples

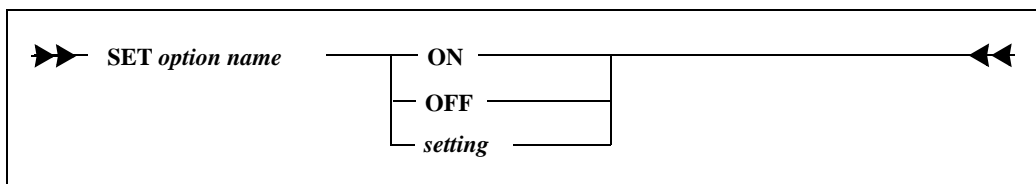
This script sets restriction mode on, establishes a set of ROWID values for a table, then saves those ROWID values for later use, ending restriction mode at the same time.

```
SET FILTER ON;
SELECT * FROM COMPANY WHERE CITY='Milano';
SAVE FILTER Italia_company;
```

## See also

SET FILTER  
ERASE FILTER

# SET



This command sets environment options for the SQLTalk session. Some options can be turned ON or OFF; others require a numeric or string constant assignment.

The SHOW command displays the setting of environment options.

## Clauses

option name

This is the name of the option that is being set. The options are described below.

### AUTOCOMMIT

If this is ON, the database is committed automatically after each SQL command. Otherwise, the database is committed only when you issue a COMMIT command. A commit via AUTOCOMMIT is identical to one issued explicitly.

Any operation performed on the cursor causes a commit for all cursors connected to the database. However, an operation on a cursor where autocommit is not turned on does *not* cause an automatic commit.

For a cursor in a distributed transaction, AUTOCOMMIT causes an implicit coordinated commit.

The AUTOCOMMIT option cannot be turned on at the same time as the BULK option. Turning on the BULK option automatically disables AUTOCOMMIT.

AUTOCOMMIT only affects the following operations:

- compile and execute
- execute
- store a command
- drop a command
- retrieve a command
- release log

For other types of operations, you should do an explicit commit, even if you have RL isolation.

There is no specific authority required to run SET AUTOCOMMIT.

The default is OFF.

### BELL

If this is ON, the bell sounds when user interaction is desired.

There is no specific authority required to run SET BELL.

The default is OFF.

## BM integer constant

This sets the bottom margin of a report (the number of blank lines that are displayed at the bottom of a page). The value can range from 0 to PAGESIZE.

There is no specific authority required to set this option.

The default is 0.

## BULK

If this is ON, operations are buffered in the output message buffer as much as possible.

The bulk execute feature reduces the network traffic for multi-row inserts, deletes, and updates, particularly across a network. In bulk execute mode, data values are buffered so that many rows can be sent to the server in one message.

Increasing the size of the output message buffer with the SET OUTMESSAGE command increases the number of operations that can be buffered in one message to the server, which improves performance. The minimum setting for this buffer is 320 bytes.

The BULK option cannot be turned on at the same time as the AUTOCOMMIT option. Turning on BULK automatically disables AUTOCOMMIT.

This setting is cursor specific.

You must have CONNECT authority to set this option.

The default is OFF.

## CHECKPOINT integer constant

This specifies how often a recovery checkpoint operation is performed. This operation is required for the automatic crash recovery mechanism.

Depending on the applications running against the server, a checkpoint operation can affect performance. If this happens, the checkpoint interval can be increased until the desired performance is attained.

The default checkpoint time interval is very small, yielding a crash recovery time of less than a minute. If a longer crash recovery time can be tolerated, the interval can be increased up to 30 minutes.

You must have DBA authority to set this option.

The default is one minute.

## CLOSETRANSMETHOD [COMMIT | ROLLBACK | DEFAULT]

This option specifies whether a COMMIT or ROLLBACK is issued before a connection handle is destroyed. The default is DEFAULT which means that this

behavior is dependent on the database server to which the user is connected. In the case of SQLBase, the DEFAULT setting issues a COMMIT before a connection handle is terminated. Refer to documentation for your specific server for other default settings.

This option also specifies whether a COMMIT or ROLLBACK is issued before disconnecting a final cursor that was implicitly connected or connected using the CONNECT command with the cursor number along with the following clause options: databasename, username, and password.

#### CURSORNAME string

This assigns a name to the current cursor. SET CURSORNAME is used before executing an UPDATE or DELETE with a CURRENT OF clause or an INSERT with an ADJUSTING clause.

The cursor name must be eight characters or less.

There is some overhead for fetches when a cursor name is assigned because the server must keep track of the current cursor position. You can de-assign a cursor name by specifying a null value. The server optimizes fetches when a cursor is not assigned.

You must have CONNECT authority to set this option.

#### DATEPICTURE 'picture'

This edits DATE data type values according to the specified print picture. See the explanation of the COLUMN command for information about print pictures.

There is no specific authority required to set this option.

#### DATETIMEPICTURE 'picture'

This edits DATETIME data type values according to the specified print picture. See the explanation of the COLUMN command for information about print pictures.

There is no specific authority required to set this option.

#### DB2

If this is ON, SQLBase is DB2 compatible.

If this is ON, you can use the FOR UPDATE OF clause in a SELECT command.

You must have CONNECT authority to set this option.

The default is OFF.

## DECHO

If this is ON, data accompanying a SQL command in a bind variable is echoed to the screen.

If this is OFF, the data is not echoed.

This command is useful when INSERTing large amounts of data.

If ECHO is OFF, DECHO cannot be set ON.

There is no specific authority required to set this option.

The default is ON.

## DEVICESIZE integer constant

This indicates the width of the output device in number of characters. Typical screens are 80 characters wide. Printers usually range from 80 to 132 characters. If the LINESIZE is less than the DEVICESIZE, excess characters to the right are truncated, unless the LEFT or RIGHT commands are used to position the displayed columns appropriately.

There is no specific authority required to set this option.

The default DEVICESIZE is 80. The range is 0 to 132.

## DISTRANS

This indicates that all subsequent commands will be part of a distributed transaction, which spans multiple databases. Currently, you cannot include a BACKUP or RESTORE command in a distributed transaction. You cannot turn DISTRANS ON if connection handles exist. The default is OFF.

---

**Note:** Connection handles are not supported for distributed transactions. Therefore, if you are using connection handles, distributed transactions cannot be enabled. For details on connection handles, read the sections in this chapter on the BEGIN CONNECTION and CONNECT commands.

---

You can set DISTRANS ON in the same connection in which it was set OFF. All connects after DISTRANS has been set OFF belong to non-distributed transactions.

There is no specific authority required to set this option.

## ECHO

If this is ON, command text is echoed to the screen when executing a command file using the RUN command or a precompiled command with the EXECUTE command.

If this is OFF, the text and any associated data is not echoed.

There is no specific authority required to set this option.

The default is ON.

**ERRORLEVEL n**

This controls the level of detail in the error messages that SQLTalk displays.

The number that you specify can be from 1 to 3. The table below explains the meaning of each number.

Error Level	Explanation
0	<p>This is the default setting. Only the error message is displayed.</p> <p>For example, if you enter the SQL command “CREATE JUNK;”, you get this message:</p> <p>Error: Invalid CREATE command</p>
1	<p>This level displays:</p> <p>The error message text.</p> <p>The error code number.</p> <p>The mnemonic.</p> <p>For example, if you enter the SQL command “CREATE JUNK;”, you get this message:</p> <p>Error: 00903 PRS ICC Invalid CREATE command</p>
2	<p>This level displays:</p> <p>The error message text.</p> <p>The error code number.</p> <p>The mnemonic.</p> <p>The error reason.</p> <p>For example, if you enter the SQL command “CREATE JUNK;”, you get this message:</p> <p>Error: 00903 PRS ICC Invalid CREATE command</p> <p>Reason: The next keyword immediately following a CREATE verb is invalid.</p>

Error Level	Explanation
3	<p>This level displays:</p> <ul style="list-style-type: none"> <li>The error message text.</li> <li>The error code number.</li> <li>The mnemonic.</li> <li>The error reason.</li> <li>The error remedy.</li> </ul> <p>For example, if you enter the SQL command “CREATE JUNK;”, you get this message:</p> <p>Error: 00903 PRS ICC Invalid CREATE command</p> <p>Reason: The next keyword immediately following a CREATE verb is invalid.</p> <p>Remedy: Verify that the keyword immediately following the CREATE verb keyword is either TABLE, INDEX, UNIQUE (index), SYNONYM, PUBLIC (synonym), or VIEW.</p>

There is no specific authority required to set this option.

## ETIME

If this is ON, SQLBase displays the time spent to generate the results, including formatting them and displaying them on the screen. A *before* system time is recorded before beginning a SQL operation (before the compile or before the retrieve of a stored command). An *after* system time is recorded when the statement has completed processing (after the execute or final fetch) and displaying all of the data on screen with the final times.

TIME is a better indicator of database performance than ETIME. TIME is the time it takes to execute each API call.

TIME is automatically turned OFF when ETIME is turned ON.

There is no specific authority required to set this option.

The default is OFF.

## EXTENSION integer constant

This option requires DBA authority.

This sets the extension size of a database file in kilobytes.

SQLBase databases grow dynamically as data is added, and they expand in units called *extensions* (extents). When a database becomes full, SQLBase must add another extent to the database. You can set the size of the extent. This option sets the extension size for both partitioned and non-partitioned databases.

In non-partitioned databases, the *.dbs* file grows in 100 kilobyte extents by default.

In partitioned databases, the default extent is one megabyte. The extension size is rounded up one megabyte from the user setting. For example, if you set the extension at 1024 kilobytes, the actual extension size is still 1024 kilobytes. However, if you set the extension to 1025 kilobytes, the actual extension is two megabytes (2048 kilobytes).

## FETCHTHROUGH

This feature is used to avoid retrieving rows from the client's input message buffer that have been updated.

When `FETCHTHROUGH` is `OFF`, SQLBase fetches rows from the client's input message buffer if possible.

When `FETCHTHROUGH` is `ON`, SQLBase fetches data from the backend, ensuring the most up-to-date data. Only one row is fetched from the back-end at a time.

---

**Note:** For procedures, if you want the *On Procedure Fetch* section to execute exactly once for every fetch call from the client (returning one row at a time) set `FETCHTHROUGH` mode on at the client (the default is off).

---

Using `FETCHTHROUGH` increases response time. Only use `FETCHTHROUGH` when it is necessary to get the most up-to-date row.

This setting is cursor specific.

You must have `CONNECT` authority to set this option.

The default is `OFF`.

SQLBase creates a virtual table with a pseudo row ID if a `SELECT` statement containing one or more of the following options creates a result set:

- `DISTINCT`
- `GROUP BY`
- `HAVING`
- `UNION`
- `ORDER BY` (if it includes a join or is on non-indexed columns)
- aggregate function
- a complex view

In these cases, rows in the result set *cannot* be mapped to the rows in the database. If you UPDATE in this situation and later fetch an UPDATED row, the row will *not* reflect the UPDATE even if FETCHTHROUGH is ON.

#### FILTER ON | OFF | *name*

Using this clause with ON or *name* is equivalent to issuing a SET SCROLL ON followed by SET RESTRICTION ON. Both result set mode and restriction mode are activated.

If *name* is used, a set of restriction-mode ROWIDs that was saved previously under *name* (see SAVE FILTER command) will be retrieved and activated.

If OFF is used, both result set mode and restriction mode are deactivated.

More detail about restriction mode is available in Chapter 12, *Result Sets*, of the *SQLBase Advanced Topics Guide*.

#### HEADING

If this is OFF, default headings are not displayed above each column of data during query output.

If this is ON, default headings are displayed above columns for which a COLUMN HEADING has been given.

Column headings specified with the COLUMN command are displayed regardless of the setting of the HEADING option. Default headings contains column names (or expressions) as they appear in the select list of a SELECT command.

There is no specific authority required to set this option.

The default is ON.

#### HISFILESIZE integer constant

This option requires DBA authority.

If the READONLY option is enabled for a database, this parameter limits the size of the read-only history file. The size is specified in kilobytes.

The default history file size is one megabyte (1000 kilobytes).

#### INDENT integer constant

This sets the number of spaces that each wrapped line in a multi-line row is indented. Usually, one or two spaces are appropriate.

The default is one.

There is no specific authority required to set this option.

If you set this to zero (0), continuation lines are not indented.

#### INMESSAGE integer constant

This sets the default maximum size (in bytes) for the SQLBase input message buffer. The input message buffer refers to input to the application (such as the result of a query).

The INMESSAGE size affects the cursor that is active when the command is given.

Most query data will not exceed the default input message buffer size, but if it does, you can use this command to raise the size of the INMESSAGE.

The input message buffer is allocated on both the client computer and on the database server. The database server builds an input message in this buffer on the database server computer and sends it across the network to a buffer of the same size on the client. It is called an *input* message buffer because it is input from the client's point of view.

There is one input message buffer per connected cursor on the client computer. The server maintains one input message buffer that is the size of the largest input message buffer on the client computer.

The input message buffer can receive a return code indicating that the specified operation was successful, the data that is being fetched, and other information. While fetching data from the database, SQLBase compacts as many rows as possible into one input message buffer.

Each FETCH reads the next row from the input message buffer until they are exhausted. At this instant, SQLBase transparently fetches the next input buffer of rows depending on the isolation level.

A large input message buffer can help performance while fetching data from the database because it reduces the number of network messages. Note that a large input message buffer can have a diverse effect on the entire system throughput because of concurrence. Any row currently in the input message buffer can have a shared lock on it (depending on the isolation level) preventing other users from changing that row. Therefore, a large input message buffer can cause more shared locks to remain than are necessary.

Read the explanation of ISOLATION (SET option) for more information about how each isolation level uses the input message buffer.

SQLBase automatically maintains an input message buffer large enough to hold at least one row of data. Despite the specified input message size, SQLBase dynamically allocates more space if necessary.

A large input message buffer helps performance when reading LONG VARCHAR columns.

You can also improve overall system performance by decreasing the size of the input message buffer when an application does not need to fetch data.

Each cursor has one input message buffer associated with it on the client.

You must have CONNECT authority to set this option.

## ISOLATION level

This sets the isolation level in a multi-user environment. The isolation level controls the effect that changes made by one user have on another user accessing the same tables. SQLBase supports these isolation levels:

- Read Repeatability (RR).
- Cursor Stability (CS).
- Read Only (RO).
- Release Locks (RL).

Choose an isolation level based on the application's requirements for consistency and concurrence.

The isolation level you set applies to *all* the cursors that the program connects to the database. It also persists across COMMIT statements.

If you change isolation levels, it causes an implicit commit for *all* cursors that the program has connected to the database.

Changing the isolation level to a different level causes an implicit commit which destroys compiled commands. However, performing the SET ISOLATION command and specifying an isolation level that is the same as the current isolation level does *not* cause an implicit commit.

You must have CONNECT authority to set this option.

### *Isolation Levels and the Input Message Buffer*

Each isolation level uses the input message buffer differently. This buffer is allocated on the client computer and the server computer. The database server builds a message and sends it to the input message buffer on the client computer. This buffer is considered "input" with respect to the client computer.

There is one input message buffer per connected cursor on the client computer. On the server, there is one input message buffer that is the size of the largest input message buffer on the client computer.

The input message buffer receives data requested by the client that has been sent by the server.

The Read Repeatability, Read Only, and Release Lock isolation levels each fill the input message buffer with as many rows as it can accommodate. The Cursor Stability isolation level sends only one row at a time to the input message buffer.

Any row in the input message buffer may have a shared lock on it depending on the isolation level setting, preventing other users from changing that row.

The table below summarizes how page locking and the input message buffer are affected by each isolation level.

Isolation Level	Shared Lock Duration and Scope
Read Repeatability	Locks are maintained for the duration of a transaction and more than one page can be locked.
Cursor Stability	Locks are maintained for the duration of a transaction, but only the current page is locked.
Read Only	No locks are maintained.
Release Lock	All locks are released by the time SQLBase returns control to the client.

**Read Repeatability (RR).** This isolation level means that all pages which you access stay locked for other users until you COMMIT your transaction. If the same data is read again during the transaction, those rows would not have changed. This guarantees that the data accessed is consistent for the life of the transaction. Identical SELECT commands will return identical rows since data cannot be changed by other users during the transaction. In this situation, other users may wait for your COMMIT command.

Read Repeatability is the default isolation level.

The Read Repeatability isolation level fills the input message buffer with rows. All shared locks remain regardless of the size of the input message buffer until the application issues COMMIT or ROLLBACK.

**Cursor Stability (CS).** This isolation level means that only the page you are processing at the moment is locked to other users. A shared lock is placed on a page for as long as the cursor is on that page (while the cursor is stable). Exclusive locks are held until a COMMIT. Shared locks, on the other hand, are only held while the cursor is stable on a page. When the cursor moves off the page and the page is no longer in the message buffer, the page’s shared lock is dropped. Other pages you

accessed during the transaction are available to other users and they do not have to wait for your COMMIT.

Data that has been read during a transaction may be changed by other users when the cursor moves to a new page.

Only one row is sent in the input message buffer under the Cursor Stability isolation level despite the size of the input message buffer. In other words, each FETCH causes the client and server to exchange messages across the network.

Use Cursor Stability when you want to update one row at a time using the CURRENT OF cursor clause. When the row is fetched to the client input message buffer, its page will have a shared lock, which means that no other transaction will be able to update it.

**Read Only (RO).** This isolation level places no locks on the database and can only be used for reading data. DML and DDL operations are *not* allowed while in Read-Only isolation. This isolation level provides a view of the data as it existed when you gave the SELECT command. If you request a page that is locked by another concurrent transaction, SQLBase provides an older copy of the page from the read-only history file. The read-only history file maintains multiple copies of database pages that have been changed.

This is an appropriate isolation level if the data wanted must be consistent but not necessarily current. This isolation level also guarantees maximum concurrence.

Read-only transactions may affect performance, so they are disabled by default. Read-only transactions can be turned on with the SET READONLY command or by specifying the *readonly* keyword in *sql.ini*.

This isolation level fills the input message buffer with rows.

The Read-Only isolation level is disabled when the READONLYDATABASE attribute is on.

**Release Lock (RL).** Under Cursor Stability, when a reader moves off a database page, the shared lock acquired when the page was read is dropped. *However*, if a row from the page is still in the message buffer, the page is still locked.

In contrast, the *Release Locks (RL)* isolation level increases concurrence by releasing all shared locks by the time control returns to the client.

When the next message or command is sent to the database, SQLBase acquires share locks on only those pages that belong the current cursor. The locks are obtained regardless of the current command type. Just before returning to the user, SQLBase releases all shared locks. It also internally notes the page numbers of those pages that had locks on them.

This isolation level fills the input message buffer with rows, which minimizes network traffic.

Use this isolation level for browsing applications which display a set of rows to a user.

**LIMIT** integer constant | OFF

This specifies the maximum number of rows that are displayed as a result of a **SELECT** command. This is useful when testing queries on a large database or where displaying a few rows is enough indication of success for that query. The number must be a positive and greater than zero.

If set to OFF, the default, the previous limit is removed.

There is no specific authority required to set this option.

**LINESIZE** integer constant

This sets the width of an output line in number of characters. If the **LINEWRAP** option is ON, **LINESIZE** can be set to be equal to or less than **DEVICESIZE** to avoid truncation of rows.

There is no specific authority required to set this option.

The default is 80. The range is 0-132.

**LINESPACE** integer constant

This sets the spacing between rows:

- Single (1)
- Double (2)
- Triple (3)

There is no specific authority required to set this option.

Rows that wrap to the next line are always at least double spaced. The default is single spacing (1).

**LINEWRAP**

If this is ON, each row is wrapped to a new line if its width exceeds the **LINESIZE**. Otherwise, the row is truncated after the last column that fits on the line. Wrapping occurs up to a maximum of 5 lines.

There is no specific authority required to set this option.

The default is OFF.

### LM integer constant

This sets the left margin of a report. The value of LM can range from 0 to LINESIZE. The default is 0. There is no specific authority required to set this option.

### LOADBUFFER integer constant

This sets the size of the load buffer used to read either ASCII or DIF files during a LOAD operation. The load buffer does not have to equal the size of the entire input buffer; it only needs to be as large as the largest row of data. It processes a minimum of one LOAD file row at time.

The default LOADBUFFER size is 1000 bytes. The minimum size for a LOAD ASCII operation is 50; a LOAD DIF requires a setting of 150.

This option requires CONNECT authority.

### LOADVERSION integer constant

This option is *not* applicable to SQLBase v6.0. If you are using a LOAD file that was created by a previous SQLBase release, this option lets you specify what version created the LOAD file.

This setting is cursor specific.

You must have CONNECT authority to set this option.

### LOGBACKUP

This option requires DBA authority.

If media recovery is important to your site, set the LOGBACKUP parameter to ON, specifying that logs are to be backed up before SQLBase deletes them.

By default, LOGBACKUP is OFF 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 should be set ON only *once*. The setting will stay active until changed. You do not need to set this each time a database is brought back on-line. Resetting this option affects whether log files are deleted or saved for archiving. To avoid gaps in your log files, set this option ON only *once*.

LOGBACKUP must be on to do a BACKUP DATABASE or a BACKUP LOGS, but does not need to be on to do a BACKUP SNAPSHOT.

The default is OFF.

## LOGFILEPREALLOC

This option requires DBA authority.

If this is OFF, the current log file grows in increments of 10% of its current size. This uses space conservatively but can result in a fragmented log file which may affect performance.

If this is ON, all log files for the database are preallocated full sized (1 megabyte) and do not "grow" as needed.

This option only needs to be turned on once for a database (such as immediately after it is created). Once you turn LOGFILEPREALLOC ON, it stays on, even if you take the server down and bring it back up later.

The default setting is OFF.

## LOGFILESIZE integer constant

This option requires DBA authority.

This sets the size of the transaction log file in kilobytes. When the current transaction log file grows to the specified size, a new transaction log file is created. A large transaction log file is best for performance because it ensures that log files are not created too often and the log files do not become fragmented, as are log files which grow in chunks. However, if the transaction log file is too large, it wastes disk space.

This option only needs to be set once for a database (such as just after it is created). Once you enter SET LOGFILESIZE, the database remembers that size even if you take the server down and bring it back up later.

The extension size is rounded up one megabyte from the user setting. For example, if you set the extension at 1024 kilobytes, the actual extension size is still 1024 kilobytes. However, if you set the extension to 1025 kilobytes, the actual extension is two megabytes (2048 kilobytes).

The default is 1 megabyte (1024 kilobytes).

## LONGINFERS

This lets you read and write long data when using front end result sets with SQLNetwork routers and gateways.

This option is cursor specific.

You must have CONNECT authority to set this option.

## NEXTLOG integer constant

This option requires DBA authority.

If LOGBACKUP is ON, this option needs to be set after an *off-line* backup is done. It tells SQLBase that an off-line backup has been made and that there are now log files eligible for deletion. The integer constant is the number of the next log to be backed up.

For example, if you make an off-line backup of *mydbs.dbs*, *1.log*, *2.log*, and *3.log*, set the NEXTLOG option to 4. SQLBase then knows that *1.log*, *2.log* and *3.log* need to be deleted (because they have been backed up), and that *4.log* and all other logs which follow need to be saved for archiving.

## NOPREBUILD

If this is OFF, result sets are prebuilt on the database server. The client must wait until the entire result set is built before it can fetch the first row.

If this is ON, result sets are not prebuilt if the client:

- Is in result set mode.
- Is in Release Locks (RL) isolation level.

The advantage of SET NOPREBUILD ON is that the client does not have to wait very long to fetch the first row. The result set is built as the data is fetched.

Note that setting NOPREBUILD to OFF does not reduce any client memory consumption. Nor does it necessarily reduce the number of shared locks set by each user; this depends on the isolation level.

The server releases shared locks before control returns to the client.

NOPREBUILD is cursor specific.

You must have CONNECT authority to set this option.

The default is OFF.

## NULLS 'string '

This substitutes null values in the output with the specified string. The string must be enclosed in *single* quotes. The string can be up to 10 characters long. If the field width is smaller than the string, the string is truncated upon display. An empty string consisting of two single quotes causes nulls to be displayed as a string of blanks. The default is blanks.

There is no specific authority required to set this option.

## OPTIMIZEDBULK

This option is similar to the BULK option, with two basic differences:

- Inserts are much faster because there is only a minimum amount of generated transaction log.

- Any error in the INSERT results in the entire transaction being rolled back, not just the command.

A typical situation that uses this option is a LOAD where you cannot set RECOVERY OFF, such as when you are loading a single table into an on-line database. In this type of situation, the transaction is rolled back when an error occurs, but you can just restart the LOAD.

This option requires CONNECT authority.

#### OPTIMIZEFIRSTFETCH

This parameter lets you set the optimization mode for a particular cursor. The cursor executes the command; all queries that are compiled or stored in this cursor inherit the optimization mode in effect. This setting overrides the value set for *optimizefirstfetch* in *sql.ini*.

If set to 0, the cursor optimizes the time it takes to return the entire result set. If set to 1, the cursor optimizes the time it takes to fetch the first row of the result set.

If *sql.ini* does not have an *optimizefirstfetch* keyword, the default setting is 0 (optimize the time it takes to return the entire result set).

Note that a command that was earlier stored, retrieved, and executed will continue to use the execution plan with which it was compiled.

#### OPTIMIZERLEVEL integer constant

This sets the optimizer techniques that SQLBase uses for the current cursor. This lets you experiment with beta optimizing techniques in new versions of SQLBase.

When this option is 1, SQLBase uses current optimizing techniques.

When this option is 2, SQLBase uses new optimizing techniques.

This option overrides the setting of the *optimizerlevel* keyword in *sql.ini*. If *sql.ini* does not have an *optimizerlevel* keyword, the default setting is 2 (use new optimizing techniques).

This setting is cursor specific.

You must have CONNECT authority to set this option.

#### OUTMESSAGE integer constant

This sets the maximum size (in bytes) for the output message buffer. The output message buffer refers to the output of a command.

If you are inserting extremely wide table rows or creating large tables, you can increase the buffer size. Usually the default is sufficient.

The output 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 of the same size on the database server. It is called an *output* message buffer because it is output from the client's point of view.

The most important messages sent from the client to the database server are SQL commands to compile or a row of data to insert.

A large output message buffer does not necessarily increase performance because it only needs to be large enough to hold the largest SQL command to compile or large enough to hold the largest row of data to insert. A large output message buffer can allocate space unnecessarily on both the client and the server. Rows are always inserted and sent one row at a time (except in bulk execute mode). A larger output message buffer does *not* reduce network traffic.

SQLBase automatically maintains an output message buffer large enough to hold any SQL command or a row to insert of any length (given available memory). Despite the specified output message buffer size, SQLBase dynamically allocates more space for the output message buffer if needed.

A large output message buffer can help performance when writing LONG VARCHAR columns.

You must have CONNECT authority to set this option.

#### PAGESIZE integer constant

This sets the size of a display page in number of lines. The number must be a positive and greater than 1. The default is 20 lines.

There is no specific authority required to set this option.

#### PARTITIONS

This enables and disables access to partitioned databases.

When this option is OFF:

- You can restore *main.dbs*.
- You cannot access partitioned databases. An exception is when you do not have a *main.dbs*. After creating the first database, SQLBase creates *main.dbs* and sets *partitions* to on (1) in *sql.ini*.

When you change this option, it changes the setting of the *partitions* keyword in *sql.ini*.

You must run SET SERVER prior to running SET PARTITIONS.

PAUSE

If this is ON, you are prompted to continue or abort after each screen of data is displayed.

If this is OFF, no pause occurs. Also, if OFF is specified, the PAUSE command is ignored while executing a command file with the RUN command.

There is no specific authority required to set this option.

The default is ON.

PLANONLY

If this is ON, SQLBase displays the execution plan for a compiled SQL command. An execution plan, also called a query plan, shows what optimization techniques or particular index SQLBase is using. To compile an execution plan, set this option to ON before issuing a PREPARE statement. The default is OFF.

You must have CONNECT authority to set this option.

When PLANONLY is ON, SELECT commands are only compiled, not executed. The corresponding execution plan is saved in the PLAN\_TABLE table.

**PLAN\_TABLE.** SQLBase creates this table automatically when you execute a SQL statement with PLANONLY set to ON. It is not a system catalog table, and users can delete it.

PLAN\_TABLE contains the following columns:

Column Name	Description
QUERYNO	Number that SQLBase assigns to the query.
OUTER_TBL	Outer table of a join.
IND_USED_O	Index used to access outer table.
INNER_TBL	Inner table of a join.
IND_USED_I	Index used to access inner table.
RESULT_TBL	Join's result table.
JOIN_METHOD	Join method.
SORT	Sort used in the join.
PLANNO	Subplan number.

Column Name	Description
JOIN_TYPE	Type of join. A join can be one of the following types: <ul style="list-style-type: none"><li>• ordinary</li><li>• anti</li><li>• outer</li></ul>
SEQUENCE_NO	Order of the join with regards to the corresponding subplan.
COMMENT	Additional information on the execution plan for complex queries. Read the end of this section for details.

In addition to storing the query plan in `PLAN_TABLE`, SQLBase displays important information about the plan on-line. The online query plan displays the following information from `PLAN_TABLE`:

- outer table
- outer table's index
- inner table
- inner table's index
- join's final result
- join method

Each line in the plan represents a *temporary* (result) table needed to process the SQL command. Generated by SQLBase, these are logical, not physical tables. They are listed in the order in which SQLBase processes them.

In the online execution plan, each index and base table name is truncated to six characters. Truncated names are suffixed with an ellipse (...).

**Joins.** Joins are listed in the order in which they are processed. The execution plan shows information such as each join's inner and outer tables, the final result table, and any indexes used.

If a table in a join is a base table, the execution plan displays the table name. If it is a view or temporary table, SQLBase assigns it a name of `TMPx`, where *x* is a number corresponding to the join order. The `TMPx` table names are in ascending order, following the join order. The last temporary table is the result of the entire join command, and is called *RESULT*. See the following sample plan for an example.

Sometimes, the process of generating a temporary table is itself divided into subprocesses. A subprocess can create a temporary table of its own. For example, assume the process to generate the temporary table *TMPI* requires several subprocesses that create temporary tables *TMPI,1* and *TMPI,2*. Each of these subprocesses represents a small piece in the overall query plan called a *subplan*.

**Online query plan examples.** The following example shows a sample online query plan:

```
SET PLANONLY ON;
SELECT PRES_NAME FROM PRESIDENT X
  WHERE STATE_BORN IN
    (SELECT STATE_NAME FROM STATE
  WHERE YEAR_ENTERED + 40 >= ANY
    (SELECT MIN(YEAR_INAUGURATED)
```

```

FROM ADMINISTRATION WHERE ADMINISTRATION.PRES_NAME
= X.PRES_NAME) );
EXECUTION PLAN:
OUTER TBL INDEX USED-O  INNER TBL INDEX USED-I  RESULT TBL JOIN METHOD
=====
ADMINI...              TMP1
STATE                  TMP2          NESTED LOOP
TMP2                   PRESIDENT      RESULT      HASH JOIN

```

The execution plan shows that SQLBase processes this command in the following sequence.

- The first row of the plan indicates that SQLBase processes the innermost subquery first, since the table referenced is ADMINISTRATION.

```

(SELECT MIN(YEAR_INAUGURATED) FROM ADMINISTRATION
 WHERE ADMINISTRATION.PRES_NAME = X.PRES_NAME)

```

TMP1

SQLBase accesses the base table ADMINISTRATION by performing a table scan; there is no index used. The qualified rows from this subquery form a temporary table called *TMP1*.

- The second row from the plan references the next subquery, which joins *TMP1* with the STATE table. The qualified rows from this subquery form another temporary table *TMP2*:

```

(SELECT STATE_NAME FROM STATE
 WHERE YEAR_ENTERED + 40 >= ANY (TMP1))

```

TMP2

This process uses a nested loop join.

- Finally, the outermost select joins *TMP2* with the PRESIDENT table to get the final query result, which is stored in the **RESULT** temporary table. This process uses a hash join.

```

SELECT PRES_NAME FROM PRESIDENT X
 WHERE STATE_BORN IN (TMP2);

```

RESULT

The following example shows another query plan:

```
SET PLANONLY ON;
SELECT DISTINCT SPJ.S#, S.S#, SPJ.QTY
FROM   SPJ,SP,S
WHERE  QTY = MAXQTY
AND    (SPJ.P# = S.P#)
AND    (SP.P#=S.P#);
```

```
EXECUTION PLAN:
OUTER TBL INDEX USED-O   INNER TBL INDEX USED-I RESULT TBL JOIN METHOD
=====
S              SPJ      SPJX      TMP1      INDEX LOOP
TMP1           SP              RESULT      HASH JOIN
```

In this plan, the temporary table *TMP1* is derived from a join of the *S* and *SPJ* tables. *SPJ* is the inner table, and is accessed via an index called *SPJX*. Since there is an index, SQLBase joins *S* and *SPJ* using an index loop method.

SQLBase then joins the temporary table (*TMP1*) to *SP* with a hash join to get the final result.

To actually execute the prepared query, issue a **PERFORM** command, then set **PLANONLY** to **OFF**.

**Result set mode query plan example.** The following example shows how to obtain a query plan in result set mode.

```
SET PLANONLY ON;
PLANONLY is ON
PREPARE SELECT * FROM SYSTABLES;
STATEMENT PREPARED
PERFORM;
```

```
EXECUTION PLAN:
OUTER TBL INDEX USED-O   INNER TBL INDEX USED-I RESULT TBL JOIN METHOD
=====
              SYSTABLES              RESULT
```

```
SET PLANONLY OFF;
PLANONLY is OFF
```

**Retrieving the query plan from PLAN\_TABLE.** The online query plan does not display all the query plan information. To see the entire plan, including comments on the execution plan, issue a query against **PLAN\_TABLE**. For example, the following is a partial view of the **PLAN-TABLE** contents for a sample SQL statement.

```
SELECT * FROM PLAN_TABLE;
```

## EXECUTION PLAN:

PLANNO	OUTER_TBL	INNER_TBL	RESULT_TBL	COMMENT
=====	=====	=====	=====	=====
1		P	TMP1	SQ: 2
2		SP	TMP3	UNION
4		P	TMP4	SQ: 5
5		SP	TMP3	UNION

**Comment.** The COMMENT column contains information to help you understand the execution plan for complex queries. Here is how to interpret the COMMENT column in the previous example:

- For PLANNO 1, the COMMENT column contains “SQ:2”. The “SQ” means that the result table (TMP1) is created from a subquery. The “2” means that the subquery itself is used in the subselect identified by PLANNO 2.
- For PLANNO 4, the COMMENT column contains “SQ:5”. The “SQ” means that the result table (TMP4) is created from a subquery. The “5” means that the result of the subquery is used in the subselect identified by PLANNO 5.
- For PLANNOs 2 and 5, “UNION” in the COMMENT column means that both plans are two subselects of a UNION view (their output produces a conceptual table called TMP3).

## PRESERVECONTEXT

If this is ON, SQLBase maintains result sets after a COMMIT (cursor-context preservation). 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. The cursor context is *not* preserved after an isolation level change or after system-initiated ROLLBACKs, such as deadlocks, timeouts, etc.

The context is preserved after a user-initiated ROLLBACK if *both* of the following are true:

- The application is in Release Locks (RL) isolation level.
- A data definition language (DDL) operation was not performed.

You set PRESERVECONTEXT for one cursor at a time. All cursors with PRESERVECONTEXT set to on have their cursor context preservation maintained after a COMMIT or ROLLBACK.

For fetch operations, locks are kept on pages required to maintain the fetch position. This can block other applications that are trying to access the same data. Also, locks can prevent other applications from doing DDL operations.

This setting is cursor specific. One cursor context can be preserved while others are destroyed. The preserved context itself can be destroyed in the following situations:

- A compile request is issued using the cursor.
- A DDL command is issued in the same transaction (which may or may not be using another cursor) and the transaction is rolled back.
- An isolation level changes, even if you use another cursor to change the isolation level.
- The transaction rolls back due to an error such as a deadlock.

Only the contexts of cursors with CCP on are maintained.

This feature can be on with or without restriction mode and result set mode.

You must have CONNECT authority to set this option.

The default setting is OFF.

If the result set was created by a SELECT command that contains DISTINCT, GROUP BY, HAVING, UNION, ORDER BY, an aggregate function, or a complex view, then SQLBase creates a virtual table. In these cases, rows in the result set *cannot* be mapped to the rows in the database. If you UPDATE in this situation and later fetch an UPDATED row, the row will *not* reflect the UPDATE even if PRESERVECONTEXT is ON.

PRINTLEVEL integer constant

This sets the level of detail for the messages on the process activity server display (0-4).

The default is 0.

You must have DBA authority to set this option.

READONLY

If this is ON, the read only isolation level can be used. The read-only isolation level gives a view of the data as it was when the transaction began. If this is ON, a read-only history file is maintained that contains multiple copies of database pages that are being modified.

This feature can be set in *sql.ini* or in a SET command. If specified in *sql.ini*, the setting applies to all databases on the server. The SET READONLY command changes the setting for the current database.

Read-only transactions may negatively affect performance, so they are disabled by default.

If you give the command SET READONLY DEFAULT, the setting in *sql.ini* is used; if not set in *sql.ini*, then the internal default is used.

You must have DBA authority to set this option.

The default is OFF.

## READONLYDATABASE

If this is ON, the database is set to read-only.

If this is OFF, the database is read-write.

Once you set a database to read-only, no update statements are allowed, and log files are disabled. You must be the only connected user to turn read-only on or off.

When READONLYDATABASE is on, the Read-Only isolation level is disabled.

You must set *tempdir* (in *sql.ini* or *autoexec.bat*) to tell SQLBase where to store temporary files. The temporary files are stored in a subdirectory of the directory pointed to by *tempdir*. The name of the subdirectory is the same as the database name.

You must have DBA authority to set this option.

The default setting is OFF.

## RECOVERY

In most situations you should leave RECOVERY turned ON. However, when RECOVERY is turned OFF, there is less file I/O (and a corresponding increase in performance) and less disk space is used. Therefore, you may want to SET RECOVERY OFF when loading large amounts of data.

You must have CONNECT authority to set this option.

When RECOVERY is OFF, transaction logging is not performed. Changes to the database before a COMMIT cannot be rolled back if the transaction fails, and the database will not be salvageable if the database is damaged by an user error, media failure, or power failure. Often, your database will corrupt if you run SET RECOVERY OFF and your application crashes.

For a multi-user database server, recovery can only be established by the first user connecting to a database, and SET RECOVERY must be the first command given in the SQLTalk session.

Since RECOVERY is a database-specific option, if that first user sets RECOVERY ON or OFF, it stays that way for all other users who connect to the same database. After RECOVERY is set OFF, all users who later try to connect to the database will get a message saying that recovery is not being performed.

If you turn recovery OFF, the following SET options will be reset to their default values:

- AUTOCOMMIT
- BULK EXECUTE
- PRESERVECONTEXT
- RESTRICTION
- ROLLBACK
- SCROLL
- ISOLATION
- LOADVERSION
- TIMEOUT

The default is ON.

## RESTRICTION

If this is ON, each query is a subquery of the previous result set. Therefore, if you are in restriction mode and your first query selects everyone living in California, and a second query selects all people with brown eyes, the second query returns only people who live in California *and* who have brown eyes. Input to each select for the current SELECT is the output of the previous SELECT. More detail about restriction mode is available in Chapter 12, *Result Sets*, of the book *SQLBase Advanced Topics Guide*.

You can only turn on restriction mode after turning on scroll mode (SET SCROLL ON). Note that restriction mode is not supported for procedures.

This setting is cursor specific.

You must have CONNECT authority to set this option.

The default is OFF.

## RM integer constant

This sets the right margin of a report. The value can range from 0 to LINESIZE. The default is 0.

There is no specific authority required to set this option.

## ROLLBACK

If this is ON, the entire transaction is rolled back when there is a lock timeout. If this is OFF, only the current command is rolled back when there is a lock timeout.

This setting is cursor specific.

You must have CONNECT authority to set this option.

The default is ON.

## SCREEN

If this is OFF, no output is displayed to the screen.

There is no specific authority required to set this option.

The default is ON.

## SCROLL

If this is ON, you can randomly position within a set of rows and begin fetching instead of always fetching from the beginning. This is called result set mode.

If you turn on result sets on the client, you can scroll forwards and backwards through the result set returned by a procedure.

The result set for procedures is preserved across COMMIT and ROLLBACK operations, even if preserve context mode is off.

You can position anywhere using the SET SCROLLROW command. Subsequent fetches can be performed with the FETCH command. Input to a SELECT in scroll mode is all the rows of the SELECTed tables, which is how SELECTs normally work outside of scroll mode.

This is similar to the SCROLL CURSOR capability in the ANSI SQL standard.

This setting is cursor specific.

You must have CONNECT authority to set this option.

The default is OFF.

---

**Note:** If you do not want to use scrollable cursors, do not turn result set mode on. SQLBase builds the result set for a procedure if result set mode is turned on; this can result in unnecessary performance degradation.

---

## SCROLLROW integer constant

This positions the scroll cursor to a specific row where the first row is relative to zero. When a subsequent FETCH command is given, the display starts at the row specified by the SET SCROLLROW command. The SET SCROLL ON command (result set mode) must have been previously given.

You must have CONNECT authority to set this option.

SPACE integer constant

This indicates the number of characters that are blank between each column in a report. The value of SPACE can range from 0 to LINESIZE. The default is 1.

There is no specific authority required to set this option.

SPANLIMIT integer constant

This option requires DBA authority.

This specifies the number of log files that an active transaction is allowed to span. As new log files are created, this limit is checked for all active transactions. Long running transactions that violate the limit are rolled back.

Long running active transactions can pin down log files that otherwise could be deleted (or backed up and deleted if LOGBACKUP is enabled). You can limit the amount of log pinned down by active transactions by specifying the transaction span limit.

By default, the transaction span limit is set to zero which disables the limit checking.

TIME

If this is ON, the results of each SQL command are followed by the elapsed time taken to obtain the results. TIME is the time it takes to perform each function call. The system time is recorded before and after each API call.

TIME is a better indicator of database performance than ETIME. TIME is the time it takes to execute each API call. ETIME is the time it takes to generate the results, including formatting them and displaying them.

There is no specific authority required to set this option.

The default is OFF.

TIMEOUT integer constant

This specifies the number of seconds to wait for a database lock to be acquired. After the specified time has elapsed, the transaction or command is rolled back (see the ROLLBACK option).

Valid timeout values are:

Table 1:

1 - 1800	Seconds to wait for a lock (from 1 second to 30 minutes).
-1	Wait forever for a lock held in an incompatible mode by another transaction (infinite timeout).

**Table 1:**

0	Never wait for a lock and immediately return a timeout error (no wait locking).
---	---

The default setting is 300 seconds.

You must have CONNECT authority to set this option.

This setting is transaction specific, like ISOLATION. If you have two cursors connecting to the same database, for example, they are in the same transaction, and cannot have different TIMEOUT settings. However, two cursors connecting to two different databases belong to two transactions, and do reserve a unique TIMEOUT setting.

The timeout is set on a per-connect basis and remains in effect until the next SET TIMEOUT command is given.

SET TIMEOUT is not meaningful in a single-user system.

#### TIMEPICTURE 'picture'

This edits TIME data type values according to the specified print picture. See the description of the COLUMN command for information about print picture.

There is no specific authority required to set this option.

#### TIMESTAMP

If this is ON, each activity log entry has a timestamp.

You must have DBA authority to enable this option.

The default is OFF.

#### TM integer constant

This sets the top margin of a report (the number of blank lines that is displayed at the top of a page). The value can range from 0 to PAGESIZE. The default is 0.

There is no specific authority required to set this option.

#### TRACE

If this is ON, statement tracing is enabled. It is cursor specific.

When tracing is enabled, SQLBase displays each line of the procedure before it executes. The output includes the SQLBase process number, the line number of each statement, and the statement being executed.

By default, statement tracing is OFF.

## TRACEFILE

If you are accessing a multi-user server, this option directs statement trace output to a file on the server or to the server's Process Activity screen. If you are accessing a single-user engine, this option directs statement trace output to the screen. This option is cursor specific.

If you have one or more TRACE statements embedded in a procedure and statement tracing is enabled as well, the output of both traces is displayed on the server's Process Activity screen. Redirecting statement trace output to a file in this situation can make it easier to read both traces.

Specify a file name and optionally, a path, to which to direct trace output. Specify OFF to direct trace output to the server's Process Activity screen.

## Examples

COMMIT after each SQL command.

```
SET AUTOCOMMIT OFF;
```

Measure and display elapsed time.

```
SET ETIME ON;
```

Suppress default column headings.

```
SET HEADING OFF;
```

Change the isolation level to Read Only.

```
SET ISOLATION RO;
```

Return only one row after a SELECT.

```
SET LIMIT 1;
```

Set the line size to 75.

```
SET LINESIZE 75;
```

Wrap wide rows.

```
SET LINEWRAP ON;
```

When a NULL is encountered in a column, display 'NA'.

```
SET NULLS 'NA' ;
```

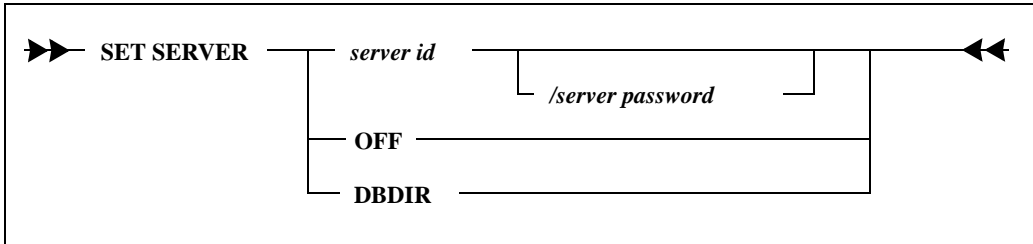
Do not pause for user confirmation of command execution.

```
SET PAUSE OFF;
```

See also

SHOW

## SET SERVER



This command establishes a server connection.

A server connection is required to perform the following administrative operations:

- Create or delete a database.
- Create a storage group.
- Install or deinstall a database.
- Drop or alter a database.
- Backup or restore a database.
- Backup or restore log files.
- Start rollforward recovery.

It is not necessary to give a SET SERVER command when using a single-user version of SQLBase.

## Clauses

### server ID

The name of the SQLBase server. Database servers are named by the *servername* keyword in *sql.ini*.

### server password

If the server password is set (with the *password* keyword in *sql.ini*), a case insensitive comparison is performed between the server password and the specified password. If a server password has not been set in *sql.ini*, it does not need to be specified.

DBDIR

This specifies the drives, paths, and directory names for the home database directory or directories. If you execute a SET SERVER DBDIR command, it changes the setting of the *dbdir* keyword in the *sql.ini* file.

OFF

A SET SERVER OFF command breaks the server connection.

Example

```
SET SERVER PROD/SECRET;  
SERVER IS SET  
...  
SET SERVER OFF;  
SERVER IS OFF
```

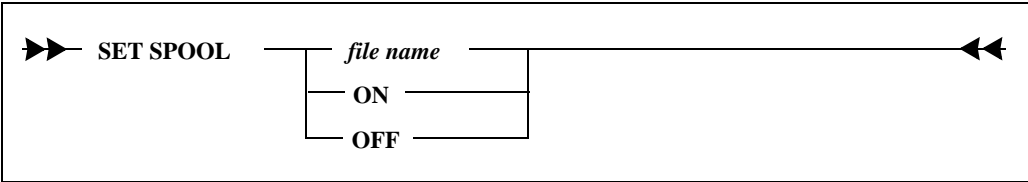
See also

BACKUP DATABASE  
RESTORE DATABASE  
SET RECOVERY

These commands are listed in the *SQL Language Reference*.

CREATE DATABASE  
DEINSTALL DATABASE  
DROP DATABASE  
INSTALL DATABASE

SET SPOOL



This command records the SQLTalk session. A spool file records all screen activity for the SQLTalk session. Spooling does not affect screen display. This file is maintained on a per user session basis.

Spooling is turned off by the SET SPOOL OFF command or when you exit from the SQLTalk session.

There is no specific authority required to set this option.

The default for spooling is OFF.

## Clauses

### file name

You can specify the name of the spool file or a default name will be assigned when you specify ON. You can specify any file name, including a drive and path.

If you start SQLTalk with the NOCONNECT option, the only way that you can turn on spooling is by executing a SET SPOOL command with a file name. SET SPOOL ON does not work when you start SQLTalk with the NOCONNECT option.

If you start spooling to the same file at a later session, you will be asked the choice of appending to or writing over the existing file.

The default spool file name is:

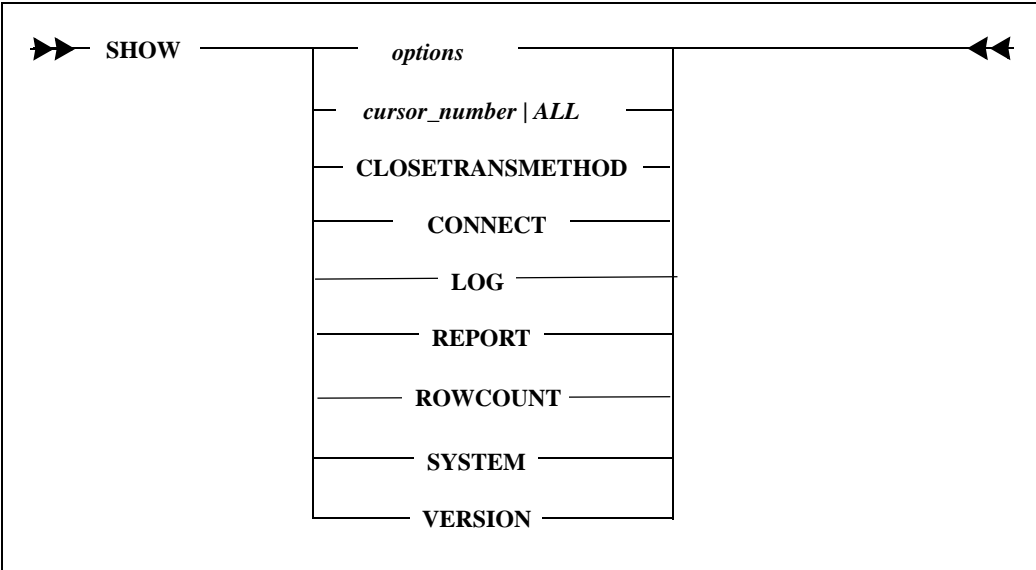
```
current directory\database name.spl
```

If the spool file already exists, you are given a choice to append to the file, overwrite the file, or abort the command.

## Examples

```
SET SPOOL ON;  
SET SPOOL OFF;  
SET SPOOL MYFILE;
```

# SHOW



This command displays information about the SQLTalk environment.

## Clauses

### options

This is the name of one of the SET options. The current value of the specified option is displayed. The **SHOW** command can also display its own options settings, which are listed below.

### *cursor-number* | *ALL*

This displays information associated with the specified cursor or all cursors in the current database. It includes the connection handle associated with each cursor and augments the information that you can obtain using the **CONNECT** option described in this section or **SHOW CONNECTION** on page 106.

### **CLOSETRANSMETHOD**

This displays whether a **COMMIT** or **ROLLBACK** is issued before closing a connection handle or disconnecting a final cursor that was implicitly connected or connected using the **CONNECT** command (which specifies the cursor number along with the following clause options: *dbname*, *username*, and *password*).

## CONNECT

This displays information associated with the current database, including cursor number, database name, and user name. The current database is that which was established at sign-on or with the most recent CONNECT or USE command.

## LOG

This displays the log parameter information.

## REPORT

This displays the current report format options. These include the SET commands (not all options are included, however), and the BREAK, BTITLE, COMPUTE and TTITLE commands.

## ROWCOUNT

This displays the number of rows in a result set. INSERTs into a result set increase the ROWCOUNT. However, DELETES, which show up as blanked-out rows in result set mode, do *not* decrease the ROWCOUNT. The deleted rows disappear on the next SELECT.

## SYSTEM

This displays configuration information, such as the heap size and whether a single-user or a multi-user system of SQLBase is being used.

## VERSION

This displays the current release version of the server software.

## Examples

Display the size of each display line.

```
SHOW LINESIZE;
```

Find out if all rows selected are being displayed.

```
SHOW LIMIT;
```

Display cursor, database and user information.

```
SHOW CONNECT;
```

Display the status of the activity log.

```
SHOW ACTIVITYLOG;
```

Display the status of the activity log timestamp.

```
SHOW TIMESTAMP;
```

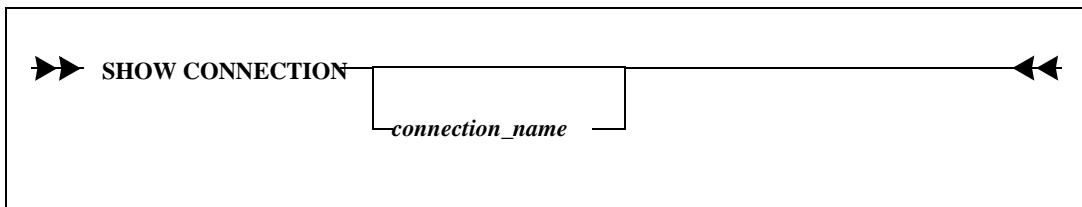
## See also

SET

## See also

COLUMN  
COMPUTE  
SHOW REPORT

# SHOW CONNECTION



This command displays information about your specified connection or the connection information for the current cursor if you do not specify a connection name. The current cursor is the one that is executing the current SQL command.

The information displayed for the connection includes the database name and the user name for the connection.

## Clauses

connection name

If desired, specify the name of the connection. If you omit the connection name, connection information is displayed for the current cursor.

## Example

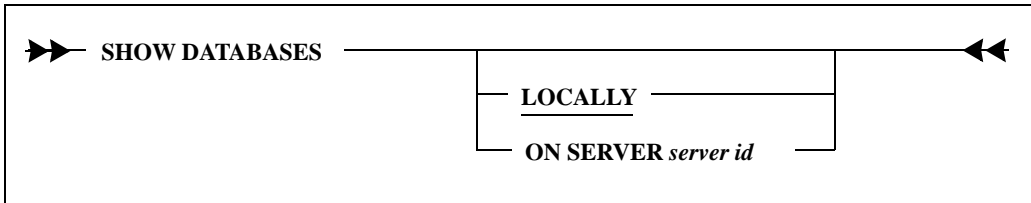
The following example displays information on connection CH2:

```
SHOW CONNECTION CH2;
```

## See also

BEGIN CONNECTION

# SHOW DATABASES



This command displays the databases available on a given server.

On a single-user system, this command displays the databases accessible on that computer. On a network, this command displays the databases installed on the network.

## Clauses

### LOCALLY

This shows all databases available on the local computer.

### ON SERVER

This displays the databases on the specified server. Database servers are named by the *servername* keyword in *sql.ini*.

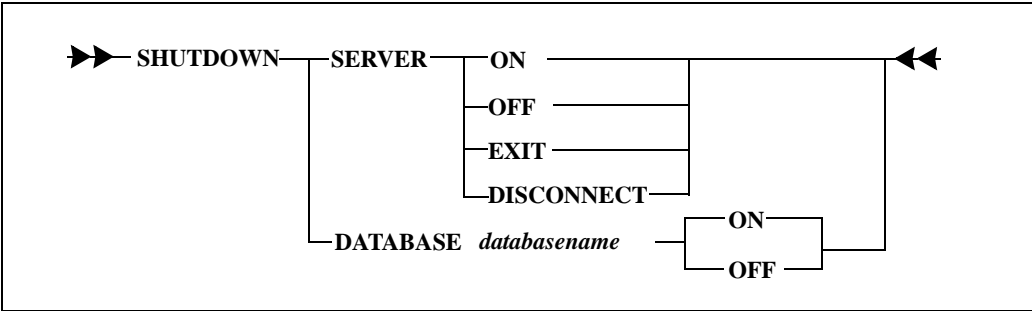
## Example

```
SHOW DATABASES ON SERVER ACCTPAY;
```

## See also

SET SERVER

# SHUTDOWN



This command shuts down SQLBase server or a database.

**Note:** Do not use the DISCONNECT clause if you are not using transaction logging.

## Clauses

### SERVER

This performs the operation at the server level.

### ON

This stops the server from accepting new connections.

Anyone trying to connect to the database receives a “shutdown server in progress” message. All current users remain connected and all current transactions continue.

### OFF

This starts accepting new connections again after you have given a SHUTDOWN SERVER ON command. You can also use this after SHUTDOWN SERVER EXIT, before the last user has disconnected, to make the server available for connections again, and to turn off the exit condition.

### EXIT

This stops accepting new connections and exits the server after the last user disconnects.

### DISCONNECT

This shuts down the server immediately. Any connected users are disconnected.

---

**Warning:** Only give this command when recovery is turned on.

---

## DATABASE

This performs the operation at the database level.

databasename

The name of the database on which to perform the operation.

## ON

This shuts down the specified database. You must be connected to the database as DBA.

Anyone (except for a user with DBA authority) trying to connect to the database receives a “shutdown in progress” message. All current users remain connected and all current transactions continue.

## OFF

This makes the database available for connections again.

## Example

This following example shuts down the server, disconnecting all active users:

```
SET SERVER servername;
SHUTDOWN SERVER EXIT;
SET SERVER OFF;
```

## SQLGET

➡➡ SQLGET ————— *parameter value* ⬅⬅

This command returns a single value, usually related to the configuration of the SQLBase server or client. It is functionally similar to the sqlget API function described in the *SQLBase API Guide*, and it uses many of the same parameter values as the API function. However, the API function processes more parameter values; the SQLGET command in SQLTalk processes only a subset of the API's parameter values.

All parameter values require that your SQLTalk session already have an existing server or database connection before they can be used successfully with this command. This is noted below.

The meaning of each parameter value listed below is explained in detail in the `sqlget` section of chapter 5 of the *SQLBase API Guide*. Briefer summaries are included here.

An example of the SQLGET command is:

```
SQLGET SQLPSINI
```

This command would return a value like `c:\program files\Gupta`, since `SQLPSINI` is the parameter value used for querying the file name and location of the server's configuration file.

Here are the *parameter values* supported by the SQLGET command:

The following parameters return a string, and require an existing server connection.

```
SQLPTPD: the directory where SQLBase places temporary files
SQLPNPF: net prefix character
SQLPNLG: net log file
SQLPERF: error code translation file
SQLPCTY: country file section
SQLPCTS: character set file name
SQLPLRD: local result set directory
SQLPALG: process activity file name
SQLPPTH: path separator on server machine
SQLPCLN: client name with internal cursor
SQLPMID: mail identification string
SQLPAID: adapter identification string
SQLPNID: network identification string
SQLPUID: user app identification string
SQLPCIS: client identification string
SQLPSINI: server configuration file location
SQLPSDIR: server module location
```

The following parameters return a string, and require an existing database connection.

```
SQLPLSS: last SQL statement
SQLPEXP: execution plan
SQLPSVN: name of server
SQLPCINI: client configuration file location
```

The following parameters return a number, and require an existing server connection.

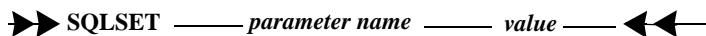
```
SQLPOOJ: Oracle outer join
SQLPNCT: netcheck algorithm
SQLPNCK: check network transmission errors
```

SQLPLCK: lock limit allocations  
 SQLPSWR: write defaults to configuration file  
 SQLPCCK: client check  
 SQLPCGR: contiguous cache pages  
 SQLPANL: apply net log (Gupta tech support only)  
 SQLPWKL: maximum work space limit  
 SQLPWKA: work space allocation unit  
 SQLPUSR: maximum number of users  
 SQLPTMO: client request timeout  
 SQLPTSS: thread stack size  
 SQLPTHM: thread mode  
 SQLPSTC: sort cache size, in pages  
 SQLPSIL: silent mode  
 SQLPROM: read-only isolation level  
 SQLPTMZ: system timezone  
 SQLPDLK: number of deadlocks  
 SQLPAPT: activate process timing  
 SQLPSTA: statistics for server  
 SQLPCTL: command time limit  
 SQLPPLV: level of process activity display  
 SQLPTMS: timestamps on process activity  
 SQLPOSR: operating system statistics sample rate  
 SQLPAWS: operating system averaging window size  
 SQLPCAC: database cache size, in KB

The following parameters return a number, and require an existing database connection.

SQLPDTL: database command time limit  
 SQLPCXP: execution plan cost  
 SQLPIMB: input message buffer size  
 SQLPOMB: output message buffer size  
 SQLPCVC: catalog version counter  
 SQLPCLG: commit-order logging

## SQLSET


**SQLSET** ——— *parameter name* ——— *value* ———

This command sets a single value, usually related to the configuration of the SQLBase server or client. It is functionally similar to the *sqlset* API function described in the *SQLBase API Guide*, and it uses many of the same parameter names

as the API function. However, the API function processes more parameter names; the `SQLSET` command in SQLTalk processes only a subset of the API's parameter names.

Some configuration values that could theoretically be handled by the `SQLSET` command are instead handled in the `SET` command. Read the documentation for that command to see these additional configuration values.

All parameter names require that your SQLTalk session already have an existing server or database connection before they can be used successfully with this command. This is noted below.

The meaning of each parameter name listed below is explained in detail in the *sqlset* section of chapter 5 of the *SQLBase API Guide*. Briefer summaries are included here.

An example of the `SQLSET` command is:

```
SQLSET SQLPCGR 60
```

This command would set the *cachegroup* size to 60 pages.

Here are the *parameter names* supported by the `SQLGET` command:

The following parameters require a string *value*, and require an existing server connection.

```
SQLPTPD: the directory where SQLBase places temporary files
SQLPNPF: net prefix character
SQLPNLG: net log file
SQLPERF: error code translation file
SQLPCTY: country file section
SQLPCTS: character set file name
SQLPLRD: local result set directory
SQLPSPF: server prefix
SQLPALG: process activity file name
SQLPPTH: path separator on server machine
SQLPCLN: client name with internal cursor
SQLPMID: mail identification string
SQLPAID: adapter identification string
SQLPNID: network identification string
SQLPUID: user app identification string
SQLPPSW: server password
```

The following parameters require a string *value*, and require an existing database connection.

```
SQLPSVN: name of server
```

The following parameters require a number *value*, and require an existing server connection.

SQLPOOJ: Oracle outer join  
 SQLPNCT: netcheck algorithm  
 SQLPNCK: check network transmission errors  
 SQLPLCK: lock limit allocations  
 SQLPSWR: write defaults to configuration file  
 SQLPCCK: client check  
 SQLPCGR: contiguous cache pages  
 SQLPANL: apply net log (Gupta tech support only)  
 SQLPWKL: maximum work space limit  
 SQLPWKA: work space allocation unit  
 SQLPUSR: maximum number of users  
 SQLPTMO: client request timeout  
 SQLPTSS: thread stack size  
 SQLPTHM: thread mode  
 SQLPSTC: sort cache size, in pages  
 SQLPSIL: silent mode  
 SQLPROM: read-only isolation level  
 SQLPTMZ: system timezone  
 SQLPAPT: activate process timing  
 SQLPSTA: statistics for server  
 SQLPCTL: command time limit  
 SQLPPLV: level of process activity display  
 SQLPTMS: timestamps on process activity  
 SQLPOSR: operating system statistics sample rate  
 SQLPAWS: operating system averaging window size  
 SQLPCAC: database cache size, in KB  
 SQLPBSS:

The following parameters require a number *value*, and require an existing database connection.

SQLPDTL: database command time limit  
 SQLPIMB: input message buffer size  
 SQLPOMB: output message buffer size  
 SQLPCLG: commit-order logging

## STORE

►► STORE auth ID. cmd/procedure name — cmd/procedure text ◄◄

This command compiles and stores a query, data manipulation command, or procedure for later execution. SQLBase stores the command's or procedure's execution plan as well, so subsequent execution is very fast.

If your data changes frequently, you should update the statistics and re-store stored procedures on a regular basis. This recompiles the stored procedures and recreates their execution plans.

You *cannot* store data definition or data control commands (such as ALTER, CREATE, DROP, or GRANT).

SQLBase keeps information about stored commands and procedures in the SYSADM.SYSCOMMANDS system table.

Other users can use stored commands by retrieving them as *user.command name*. For example, if you stored commands as SYSADM, other users can retrieve them as *SYSADM.command*. The users must have access privileges to the tables involved.

Stored commands and procedures remain in the database until you drop them with ERASE.

If you create a procedure with PROCEDURE and specify it as static, you must store the procedure with this command before you execute it.

## Clauses

auth ID.

If you are not the creator of the command or procedure, specify the authorization ID of the owner.

cmd/procedure name

The name of the command or procedure to store. If you specify a name that already exists, SQLBase prompts you for confirmation to overwrite it.

cmd/procedure text

The text of the command or procedure to compile and store. You can include bind variables in the command or procedure, but not input data. You supply input data at execution time; data binding takes place when SQLBase executes the command or procedure.

## Example

This following example compiles and stores the command MYQUERY:

```
STORE myquery
  SELECT * FROM EMP WHERE DEPTNO = :1;
```

The next example compiles and stores a command called ADDNAMES. The EXECUTE command contains bind variables, so you must enter data.

```
STORE ADDNAMES INSERT INTO FRIENDS (NAME) VALUES (:1);
EXECUTE ADDNAMES
\
PROCESSING DATA
LEN
MARGE
BETTY
/
3 Rows Inserted
```

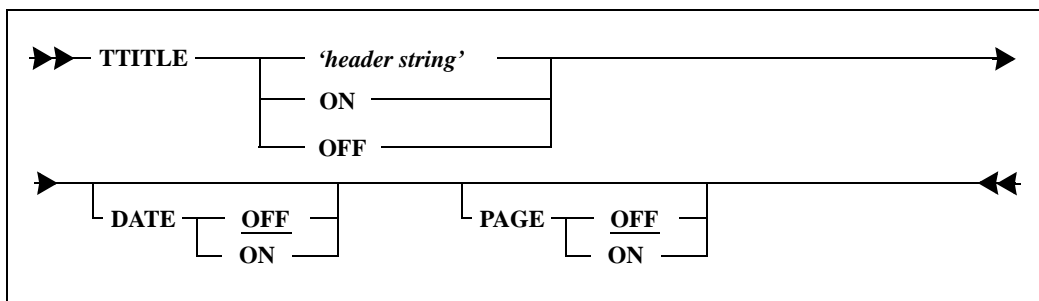
The following command stores a procedure.

```
STORE PRESPROC
Procedure PR_PRES
Local Variables
  Sql Handle Curl
Actions
  Call SqlConnection (Curl)
  Call SqlStore (Curl, 'presname', 'Select pres_name from \
sysadm.president')
  Call SqlDisconnect (Curl);
```

See also

ERASE  
EXECUTE

## TTITLE



This command displays a top title on each page of a report. Once you have entered this command, the results of all subsequent SELECT commands are displayed with the top title.

You can change the top title by entering another TTITLE command, or you can temporarily turn off TTITLE by entering the OFF command. You can turn TTITLE on again with the ON command.

Clauses

'header-string'

The header string must be enclosed within *single* quotes.

The string can consist of up to three separate substrings, each delimited by the vertical bar ( | ) character. The | represents a new-line character that causes the subsequent substring to be displayed on a new line.

The lines of the title are automatically centered. The display width of the page is specified by the SET LINESIZE command.

DATE

If this is ON, the current date is displayed in the top left-hand corner of each page of the report. The date is displayed in *Mon dd, yyyy* format (such as April 25, 1996). It is always displayed on the first line of the page, regardless of the number of lines in the title.

PAGE

If this is ON, the current page number is displayed on the top right hand corner of each page of the report. The display format is Page n (such as Page 11). It is always displayed on the first line of the page.

Example

Shown below is a TTITLE command and the resulting title.

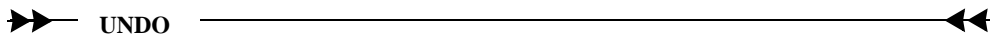
```
TTITLE 'EMPLOYEE LIST|as of December 1996'DATE
ON PAGE ON;
```

EMPLOYEE LIST		
April 23, 1996	as of December 1996	Page 1

See also

- BTITLE
- SET LINESIZE
- SHOW REPORT

# UNDO



This command is used in restriction mode to undo the current result set and return the result set to the state it was in before the last SELECT. You cannot issue UNDO multiple times to return to even earlier result set states; it can only be issued once.

More detail about restriction mode is available in Chapter 12, *Result Sets*, of the *SQLBase Advanced Topics Guide*.

## Example

```
SET SCROLL ON;
SET RESTRICTION ON;
SELECT PRES_NAME, STATE_BORN, PARTY FROM
    PRESIDENT WHERE STATE_BORN = 'VIRGINIA';
PRES_NAME      STATE_BORN PARTY
=====
Washington G   Virginia Federalist
Jefferson T    Virginia Demo-Rep
Madison J      Virginia Demo-Rep
Monroe J       Virginia Demo-Rep
Harrison W H   Virginia Whig
Tyler J        Virginia Whig
Taylor Z       Virginia Whig
Wilson W       Virginia Democratic
8 ROWS SELECTED
SELECT PRES_NAME, STATE_BORN, PARTY FROM
    PRESIDENT WHERE PARTY = 'Whig';
PRES_NAME      STATE_BORN PARTY
=====
Harrison W H   Virginia Whig
Tyler J        Virginia Whig
Taylor Z       Virginia Whig
3 ROWS SELECTED
UNDO;
UNDO COMPLETED
SELECT PRES_NAME, STATE_BORN, PARTY FROM PRESIDENT;
PRES_NAME      STATE_BORN PARTY
=====
```


```
Washington G    Virginia    Federalist
Jefferson T     Virginia    Demo-Rep
Madison J       Virginia    Demo-Rep
Monroe J        Virginia    Demo-Rep
Harrison W H    Virginia    Whig
Tyler J         Virginia    Whig
Taylor Z        Virginia    Whig
Wilson W        Virginia    Democratic

8 ROWS SELECTED
```

See also

SET FILTER  
SET RESTRICTION  
SET SCROLL

# USE

 **USE *cursor number*** 

This command establishes a new current cursor. You can use this command to switch from one cursor to another in a multi-cursor transaction after a CONNECT has been established for each cursor.

Once a CONNECT authorizes a cursor, you can switch to it later without further authorization.

## Clauses

`cursor number`

This indicates the cursor that will become the current cursor. You must have previously established a connection to the specified cursor with the CONNECT command.

## Examples

You establish a connection to a second authorization ID, using cursor 2. After you enter some commands using that cursor, cursor 1 is activated with the USE command.

```
CONNECT 2 TRUDY/X;
CURSOR 2 CONNECTED TO DEMO
CREATE TABLE X(A INT);
TABLE CREATED
INSERT INTO X VALUES (:1)
\
1
2
/

2 ROWS INSERTED


SELECT * FROM X;
A
===
1
2
2 ROWS SELECTED


USE 1;
SELECT * FROM X;
SELECT * FROM X
      ^
ERROR: TABLE HAS NOT BEEN CREATED


USE 2;
DROP TABLE X;
TABLE DROPPED
```

See also

CONNECT



## Chapter 3

# SQLTalk Reserved Words

---

This chapter lists the SQLTalk reserved words.

## SQLTalk reserved words

The following words are reserved in SQLTalk.

You can use a reserved word as an identifier if it is enclosed in double quotes, but this is *not* recommended.

ACTIVITYLOG	ADJUST
ALIAS	ALL
ALTER	APPLY
ASCII	AUTOCOMMIT
AVG	BACKUP
BACKWARD	BELL
BM	BRAND
BREAK	BTITLE
BULK	CACHE
CENTER	CHECK
CHECKPOINT	CLIENT
COLUMN	COMMANDSIZE
COMMIT	COMPRESSION
COMPUTE	CONNECT
CONTINUE	COPY
COUNT	CREATE
CURSOR	CURSORNAME
DATA	DATABASE
DATABASES	DATE
DATEPICTURE	DATETIMEPICTURE
DB2	DBAREA
DBDIR	DECHO
DEFAULT	DEINSTALL
DELETE	DESCRIBEINFO
DETECT	DEVICESIZE
DIF	DISCONNECT
DONE	DROP
DUP	ECHO
EDIT	END
ERASE	ERRORLEVEL
ERRORS	ETIME
ETONE	EXECUTE
EXIT	EXPLAIN
EXTENSION	FETCH
FETCHTHROUGH	FILTER
FILTERROW	FOREIGN
FROM	GET
GROUPCOMMIT	GROUPCOMMITDELAY
HEADING	HISFILESIZE
INDENT	INMESSAGE
INSTALL	ISOLATION
JOURNAL	LABEL
LAST	LEFT
LIMIT	LINESIZE
LINESPACE	LINEWRAP
LIST	LM
LOAD	LOADBUFFER
LOADVERSION	LOCAL
LOCALLY	LOG

LOGBACKUP	LOGFILEPREALLOC
LOGFILESIZE	LOGOFFSET
LOGS	LONGINFERS
LONGLENGTH	LONGOFFSET
MAPERROR	MAX
MIN	NEW
NEWDB	NEWPAGE
NEXTLOG	NO
NODE	NOPREBUILD
NULLS	OF
OFF	ON
OPTIMIZEDBULK	OPTIMIZEFIRSTFETCH
OUTMESSAGE	OPTIMIZERLEVEL
PAGE	OVER
PARTITIONS	PAGESIZE
PERFORM	PAUSE
PLANONLY	PICTURE
PRESERVECONTEXT	PREPARE
PRINTLEVEL	PRINT
READONLY	PUT
RECOVERY	READONLYDATABASE
RELEASE	REFINTCHECK
REORGANIZE	REMARK
RESTORE	REPORT
RETRIEVE	RESTRICTION
RM	RIGHT
ROLLFORWARD	ROLLBACK
ROWCOUNT	WRAP
ROWID	RUN
SAVE	SCHEMA
SCREEN	SCROLL
SCROLLROW	SELECTLISTNAME
SERVER	SET
SHOW	SNAPSHOT
SORTSPACE	SPACE
SPANLIMIT	SPOOL
SQL	SQLBASE
SQLGET	SQLGSI
SQLSAB	SQLSCL
SQLSET	STOGROUP
STORE	SUBTITLE
SUM	SYSTEM
TCOUNT	TIME
TIMEOUT	TIMEPICTURE
TIMESTAMP	TM
TO	TPONE
TTITLE	UNDO
UNLOAD	USE
VERSION	WIDTH
WKS	WORK



# 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.

**CRC (Cyclic Redundancy Check)**—A technique that makes unauthorized changes to a database page detectable. SQLBase appends an extra bit sequence to every database page called a Frame Check Sequence (FCS) that holds redundant information about the page. When SQLBase reads a database page, it checks the FCS to detect unauthorized changes.

**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 *sqlcncr*, 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 or engine.

***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.

***digital signature***—A unique binary number generated by an algorithm that identifies the content of a larger block of bytes.

***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.

***encryption***—The transformation of data into a form unreadable by anyone without a decryption key or password. Encryption ensures privacy by keeping information hidden from anyone for whom it is not intended, even those who can see the encrypted data. Unencrypted data is called plain text; encrypted data is called cipher text.

***engine***—See database server.

***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.

***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).

***JDBC (Java Database Connectivity)***—An industry standard for database-independent connectivity between Java and a range of databases. The JDBC provides a call-level API for SQL-based database access.

***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**—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.

**SHA (Secure Hash Algorithm)**—A hash algorithm published by the United States government that SQLBase uses to detect unauthorized changes to a database page. SHA produces a condensed representation of a database page called a message digest that is used to generate a digital signature. When SQLBase reads a page encrypted with SHA, it verifies the signature. Any unauthorized changes to the page results in a different message digest and the signature will fail to verify. It is extremely unlikely to find a page that corresponds to a given message digest, or to find two different pages which produce the same message digest.

**shared cursor**—A handle that is used by two or more Windows 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

**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.

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

---

## Symbols

\$DATA

RUN 64

## A

access path 48

ACTIVITYLOG

(reserved word) 2

ADJUST

(reserved word) 2

COLUMN 24

adjust default setting 24

ADJUSTING 15

ALIAS

(reserved word) 2

COLUMN 20

ALL

(reserved word) 2

ALTER

(reserved word) 2

ALTER COMMAND 2, 4, 114

ALTER DATABASE 5

ALTER DBAREA 5

ALTER DBSECURITY 3, 2, 5

ALTER EXPORTKEY 3, 2, 9

ALTER EXTERNAL FUNCTION 5

ALTER PASSWORD 5

ALTER STOGROUP 5

ALTER TABLE 5

ALTER TABLE (error messages) 5

ALTER TABLE (referential integrity) 5

ALTER TRIGGER 5

AM 23

APPLY

(reserved word) 2

ASCII

(reserved word) 2

AUDIT MESSAGE 5

auth ID/pwd

CONNECT 31

AUTOCOMMIT

reserved word 2

SET 70

AUTORECOMPILE 2, 4

AVG

COMPUTE 26, 28

reserved word 2

## B

backslash continuation character ( ) 11

BACKUP 3, 2, 10

DATABASE 12

FROM database 13

LOGS 12

ON CLIENT 13

ON SERVER 13

reserved word 2

SNAPSHOT 12

TO 13

BACKWARD

reserved word 2

BAT (batch option) 12

BEGIN CONNECTION 3, 2, 14

command 14

BELL

reserved word 2

SET 70

bind

data

procedure 47

variables 47

bind variable 19

examples 20

BM

reserved word 2

SET 71

BRAND

reserved word 2

BREAK 3, 2, 16

column 16

ON/OFF 16

REPORT 16

reserved word 2

BTITLE 3, 2, 17

DATE 18

footer string 18

PAGE 18

reserved word 2

BULK

reserved word 2

---

SET 71

**C**

CACHE  
     reserved word 2

case sensitivity 10

CENTER  
     reserved word 2

character adjust setting 24

character string  
     with bind variable 20

CHECK  
     reserved word 2

CHECK DATABASE 5

CHECK INDEX 5

CHECK TABLE 5

CHECKPOINT  
     reserved word 2  
     SET 71

CLIENT  
     reserved word 2

CLOSETRANSMETHOD  
     SET 71

COLUMN 3, 2, 19  
     ADJUST 24  
     ALIAS 20  
     column name 20  
     date/time picture 22  
     DUP 25  
     HEADING 20  
     NEWPAGE 25  
     NULLS 24  
     numeric picture 21  
     PICTURE 21  
     picture character 21, 23  
     PRINT 25  
     reserved word 2  
     SUBTITLE 25  
     WIDTH 20  
     WRAP 25

column  
     BREAK 16

column ID  
     COLUMN 20  
     COMPUTE 28

command  
     compile 47  
     connecting to database 14  
     parse 47  
     Prepare 47  
     repeat 10  
     RUN 63  
     stored  
         execute 40  
         owner 114  
         restriction mode 40, 57  
         result sets 40  
     terminating connection to database 37

command categories 2

command files 4

database administration 3

for storing commands/procedures 4

precompiled commands 4

report writing 3

session control 3

command file  
     PRINT 49  
     SQL 67

command files commands 4

commands  
     stored 19

COMMANDSIZE  
     reserved word 2

COMMENT 89

COMMENT ON 5

COMMIT 5  
     connection handles 71  
     reserved word 2

compile  
     command 47  
     procedure 47, 114  
     SQL command 114

COMPRESSION  
     reserved word 2

COMPUTE 3, 2, 26  
     AVG 26, 28  
     break list 28  
     COUNT 27, 28  
     MAX 27  
     MIN 26, 27  
     OFF 27  
     reserved word 2  
     SUM 26, 27

CONNECT 3, 16, 2, 30  
     auth ID 31  
     cursor 31

---

- database 31
- reserved word 2
- SHOW 105
- connecting to 14
- connection handle
  - committing transactions 71
  - rollback 71
  - terminating connections 71
- CONTINUE
  - reserved word 2
- ROLLFORWARD 62
- COPY 3, 2, 33
  - cursor 33
  - destination table 33
  - reserved word 2
  - resource table 33
- COUNT
  - COMPUTE 27, 28
  - reserved word 2
- CRC 8
- CREATE
  - reserved word 2
- CREATE COMMAND 114
- CREATE DATABASE 5
- CREATE DBAREA 5
- CREATE EXTERNAL FUNCTION 5
- CREATE INDEX 5
- CREATE STOGROUP 5
- CREATE SYNONYM 5
- CREATE TABLE 5
- CREATE TRIGGER 5
- CREATE VIEW 5
- CURSOR
  - reserved word 2
- cursor
  - assign name 72
  - CONNECT 31
  - COPY 33
  - DISCONNECT 36
  - scrollable 14
  - USE 118
- Cursor Stability (CS)
  - SET 80
- CURSORNAME
  - reserved word 2
  - SET 72
- cursors 15
  - multiple 15

- using multiple 15

cyclic redundancy check 8

## D

- DATA
  - reserved word 2
- data
  - access path 48
  - bind
    - procedure 47
    - variables 47
  - marking input 11
- DATABASE
  - BACKUP 12
  - reserved word 2
  - RESTORE 55
- database 14, 37, 106
  - backup 10
  - CONNECT 31
  - encryption 5, 7
  - how SQLBase determines 10
  - name 10
  - page alteration protection 5, 7
  - restore 55
  - roll forward 61
  - show 3, 107
  - snapshot 12
- database administration commands 3
- database audit
  - message 5
- DATABASES
  - reserved word 2
- DATE
  - BTITLE 18
  - reserved word 2
  - TTITLE 116
- date/time
  - adjust setting 25
- date/time picture
  - COLUMN 22
- DATEPICTURE
  - reserved word 2
  - SET 72
- DATETIMEPICTURE
  - reserved word 2
  - SET 72
- DB2
  - reserved word 2

---

SET 72  
DBAREA  
    reserved word 2  
DBATTRIBUTE 5  
DBDIR  
    reserved word 2  
    SET SERVER 102  
DBERROR 4, 2  
    error code 34  
DD 23  
DECHO  
    reserved word 2  
    SET 73  
decimal  
    adjust setting 24  
DEFAULT  
    reserved word 2  
defaultdatabase 10  
defaultpassword 10  
defaultuser 10  
DEINSTALL  
    reserved word 2  
DEINSTALL DATABASE 5  
DELETE 5, 14  
    reserved word 2  
delimited identifiers 10  
delimiter 10  
DESCRIBEINFO  
    reserved word 2  
destination table  
    COPY 33  
DETECT  
    reserved word 2  
detect  
    syntax error 47  
DETECT ERRORS  
    RUN 64  
determine  
    execution plan 48  
DEVICESIZE 73  
    reserved word 2  
DIF  
    reserved word 2  
DISCONNECT 3, 2, 34  
    cursor 36  
    reserved word 2  
displaying  
    connection to database 106

displaying connection 106  
DISTINCT 19  
DISTTRANS  
    SET 73  
DONE  
    reserved word 2  
double quotes  
    with bind variables 20  
double slash(//) 21  
DROP  
    reserved word 2  
DROP COMMAND 114  
DROP DATABASE 5  
DROP DBAREA 5  
DROP EXTERNAL FUNCTION 5  
DROP INDEX 5  
DROP STOGROUP 5  
DROP SYNONYM 5  
DROP TABLE 6  
DROP TRIGGER 6  
DROP VIEW 6  
DUP  
    COLUMN 25  
    reserved word 2

## E

ECHO  
    reserved word 2  
    SET 73  
EDIT 5, 2, 37  
    reserved word 2  
encryption 5, 7  
END  
    reserved word 2  
    ROLLFORWARD 62  
END CONNECTION 3, 2, 37  
    command 37  
environment control 4  
ERASE 4, 2, 38  
    example 38  
    privileges 38  
    reserved word 2  
    stored command 38  
ERASE FILTER 39  
error code  
    DBERROR 34  
error level  
    set 24

---

error messages 23  
    displaying 24  
error.sql 23  
ERRORLEVEL  
    reserved word 2  
    SET 74  
ERRORS  
    reserved word 2  
ETIME  
    reserved word 2  
    SET 75  
ETONE  
    reserved word 2  
EXECUTE 4, 2, 40  
    privileges 40  
    RECOMPILE 40  
    reserved word 2  
    stored command 40  
    stored procedure 40  
EXECUTE RECOMPILE 40  
execution plan 88  
    determine 48  
EXIT 3, 2, 42  
    reserved word 2  
EXPLAIN  
    reserved word 2  
EXTENSION  
    reserved word 2  
    SET 75

**F**  
FETCH 3, 14, 2, 43  
    PREPARE 47  
    reserved word 2  
Fetch  
    Perform 47  
FETCHTHROUGH  
    reserved word 2  
    SET 76  
FILTER 39, 68, 77  
    reserved word 2  
FILTERROW  
    reserved word 2  
footer string  
    BTITLE 18  
FOREIGN  
    reserved word 2  
forward slash (/) 10

FROM  
    reserved word 2  
FROM database name  
    BACKUP 13

**G**  
GET  
    reserved word 2  
GRANT (database authority) 6  
GRANT (table privileges) 6  
GRANT COMMAND 114  
GRANT EXECUTE ON 6  
GROUP BY 19  
GROUPCOMMIT  
    reserved word 2  
GROUPCOMMITDELAY  
    reserved word 2

**H**  
HAVING 19  
HEADING  
    COLUMN 20  
    reserved word 2  
    SET 77  
HH 23  
HISFILESIZE  
    reserved word 2  
    SET 77

**I**  
IND\_USED\_I 88  
IND\_USED\_O 88  
INDENT  
    reserved word 2  
    SET 77  
initialization file 11  
INMESSAGE  
    reserved word 2  
    SET 78  
INNER\_TBL 88  
INSERT 6, 14  
INSTALL  
    reserved word 2  
INSTALL DATABASE 6  
integer 43  
    adjust setting 24  
    FETCH 43  
ISOLATION

---

reserved word 2  
SET 79  
isolation level 79

**J**

join 18, 90, 93  
with execution plan 90  
JOIN\_METHOD 88  
JOIN\_TYPE 89  
joins 18  
JOURNAL  
reserved word 2

**L**

LABEL 6  
reserved word 2  
LAST  
reserved word 2  
LEFT 3, 44  
number of columns 44  
reserved word 2  
LIMIT  
reserved word 2  
SET 82  
LINESIZE  
reserved word 2  
SET 82  
LINESPACE  
reserved word 2  
SET 82  
LINEWRAP  
reserved word 2  
SET 82  
LIST 5, 3, 45  
reserved word 2  
LM  
reserved word 2  
SET 83  
LOAD 6  
reserved word 2  
LOADBUFFER  
reserved word 2  
SET 83  
LOADVERSION  
reserved word 2  
SET 83  
LOCAL  
reserved word 2

LOCALLY  
reserved word 2  
SHOW DATABASES 107  
LOCK DATABASE 6  
LOG  
reserved word 2  
SHOW 105  
log  
back up 12  
release 50  
restore 55  
LOGBACKUP  
reserved word 3  
SET 83  
LOGFILEPREALLOC  
reserved word 3  
SET 84  
LOGFILESIZE  
reserved word 3  
SET 84  
LOGOFFSET  
reserved word 3  
LOGS 55  
BACKUP 12  
reserved word 3  
RESTORE 55  
LONG VARCHAR  
entering in SQLTalk 21  
LONGINFERS  
reserved word 3  
SET 84  
LONGLENGTH  
reserved word 3  
LONGOFFSET  
reserved word 3

**M**

MAPERROR  
reserved word 3  
marking input data 11  
MAX  
COMPUTE 27  
reserved word 3  
message.sql 23  
MI 23  
MIN  
COMPUTE 26, 27  
reserved word 3

---

MM 23  
MON 23

## N

names 10  
NEW  
    reserved word 3  
NEWDB  
    reserved word 3  
NEWPAGE  
    COLUMN 25  
    reserved word 3  
NEXTLOG  
    reserved word 3  
    SET 84  
NO  
    reserved word 3  
NOCONNECT option 11  
NODE  
    reserved word 3  
NOPREBUILD  
    reserved word 3  
    SET 85  
NULLS  
    COLUMN 24  
    reserved word 3  
    SET 85  
numeric picture  
    COLUMN 21

## O

object  
    verify 47  
ODBC 12  
OF  
    reserved word 3  
OF column ID  
    COMPUTE 28  
OFF  
    COMPUTE 27  
    reserved word 3  
ON  
    reserved word 3  
ON break list  
    COMPUTE 28  
ON CLIENT  
    BACKUP 13  
    RESTORE 56

ON SERVER  
    BACKUP 13  
    RESTORE 56  
    SHOW DATABASES 107  
ON/OFF  
    BREAK 16  
Open DataBase Connectivity  
    see ODBC 12  
OPTIMIZEDBULK  
    reserved word 3  
    SET 85  
OPTIMIZEFIRSTFETCH  
    reserved word 3  
    SET 86  
OPTIMIZERLEVEL  
    reserved word 3  
    SET 86  
ORDER BY 19  
OUTER\_TBL 88  
OUTMESSAGE 86  
    reserved word 3  
    SET 86  
output file  
    PRINT 49  
OVER  
    reserved word 3

## P

PAGE  
    BTITLE 18  
    reserved word 3  
    TTITLE 116  
page alteration protection 5  
PAGESIZE 87  
    reserved word 3  
    SET 87  
parse  
    command 47  
PARTITIONS  
    reserved word 3  
    SET 87  
password 10  
    how SQLBase determines 10  
path  
    access 48  
PAUSE 4, 3, 46  
    reserved word 3  
    SET 88

---

PERFORM 4, 3, 46  
    example 47  
    Fetch 47  
    PREPARE 46  
    reserved word 3  
PICTURE  
    COLUMN 21  
    reserved word 3  
picture character  
    COLUMN 21, 23  
plan  
    execution  
        determine 48  
PLAN\_TABLE 88  
    columns 90  
PLANNO 88  
PLANONLY  
    reserved word 3  
    SET 88  
PM 23  
precompiling commands 4  
PREPARE 4, 3, 47, 57  
    command 47  
    Fetch 47  
    PERFORM 46  
    procedure 46, 47  
    reserved word 3  
    with execution plan 88  
PRESERVECONTEXT  
    reserved word 3  
    SET 93  
PRINT 3, 49  
    COLUMN 25  
    command file 49  
    output file 49  
    reserved word 3  
PRINTLEVEL  
    reserved word 3  
    SET 94  
PROCEDURE 6  
procedure  
    authorization ID 114  
    bind data 47  
    compile 47, 114  
    owner 114  
    Perform  
        Fetch 47  
    PREPARE 46

    Prepare 47  
        Fetch 47  
    Process Activity 100  
    run 46  
    store 114  
        limitations 114  
    stored  
        execute 40  
        restriction mode 40  
        result sets 40  
    trace  
        output 100  
    trace statements 99  
    tracing 99  
Process Activity  
    procedure 100  
PUT  
    reserved word 3

## Q

query plan 88  
QUERYNO 88  
quotes 10, 20

## R

Read Only (RO)  
    SET 81  
Read Repeatability (RR)  
    SET 80  
READONLY  
    reserved word 3  
    SET 94  
READONLYDATABASE  
    reserved word 3  
    SET 95  
RECOMPILE  
    EXECUTE 40  
recompile 4  
recompile stored commands 40  
RECOVERY  
    reserved word 3  
    SET 95  
REFINTCHECK  
    reserved word 3  
RELEASE  
    reserved word 3  
Release Lock (RL)  
    SET 81

---

RELEASE LOG 3, 50  
 REMARK 4, 3, 51  
     reserved word 3  
 REORGANIZE 3, 51  
     reserved word 3  
 repeating commands 10  
 REPORT  
     BREAK 16  
     reserved word 3  
     SHOW 105  
     SQL 67  
 report writing commands 3  
 reserved words 2  
 RESTORE 3, 53, 55  
     DATABASE 55  
     ON CLIENT 56  
     ON SERVER 56  
     reserved word 3  
     SNAPSHOT 55  
 RESTRICTION  
     reserved word 3  
     SET 96  
 restriction mode 16, 18  
     limitations 19  
     stored command 40  
     stored procedure 40  
     with joins 18  
 result set mode 13  
 result sets  
     stored procedure 40  
 result table 13  
 RESULT\_TBL 88  
 results sets  
     stored command 40  
 RETRIEVE 4, 3, 57  
     reserved word 3  
 REVOKE (database authority) 6  
 REVOKE (table privileges) 6  
 REVOKE EXECUTE ON 6  
 RIGHT 3, 59  
     reserved word 3  
 RM  
     reserved word 3  
     SET 96  
 ROLLBACK 6  
     connection handle 71  
     reserved word 3  
     SET 96  
 ROLLFORWARD 3, 61  
     CONTINUE 62  
     END 62  
     reserved word 3  
     TO BACKUP 62  
     TO END 61  
     TO TIME 62  
 row  
     counting 6  
 ROWCOUNT 6  
     SHOW 105  
 ROWID  
     reserved word 3  
 RUN 4, 3, 63  
     \$DATA 64  
     command file 63  
     DETECT ERRORS 64  
     reserved word 3  
 run  
     procedure 46

## S

SAVE 4, 3, 67  
     reserved word 3  
 SAVE FILTER 3, 68  
 SAVEPOINT 6  
 SCHEMA  
     reserved word 3  
 SCREEN  
     reserved word 3  
     SET 97  
 SCROLL  
     reserved word 3  
     SET 97  
 scrollable cursor 14  
 SCROLLROW  
     reserved word 3  
     SET 97  
 secure hash algorithm 8  
 security 47  
     ALTER DBSECURITY 2, 5  
     ALTER EXPORTKEY 2, 9  
     database page alteration protection 5, 7  
     database page encryption 5, 7  
 SELECT 6  
 Select  
     Perform  
     Fetch 47

---

Prepare  
     Fetch 47  
 SELECTLISTNAME  
     reserved word 3  
 semicolon 10  
 SEQUENCE\_NO 89  
 SERVER  
     reserved word 3  
 session control commands 3  
 SET 4, 3, 69  
     AUTOCOMMIT 70  
     BELL 70  
     BM 71  
     BULK 71  
     CHECKPOINT 71  
     CLOSETRANSMETHOD 71  
     cursor stability 80  
     CURSORNAME 72  
     DATEPICTURE 72  
     DATETIMEPICTURE 72  
     DB2 72  
     DECHO 73  
     DISTRANS 73  
     ECHO 73  
     ERRORLEVEL 74  
     ETIME 75  
     EXTENSION 75  
     FETCHTHROUGH 76  
     HEADING 77  
     HISFILESIZE 77  
     IDENT 77  
     INMESSAGE 78  
     ISOLATION 79  
     LIMIT 82  
     LINE SIZE 82  
     LINE SPACE 82  
     LINE WRAP 82  
     LM 83  
     LOADBUFFER 83  
     LOADVERSION 83  
     LOGBACKUP 83  
     LOGFILEPREALLOC 84  
     LOGFILESIZE 84  
     LONGINFERS 84  
     NEXTLOG 84  
     NOPREBUILD 85  
     NULLS 85  
     OPTIMIZEDBULK 85  
     OPTIMIZEFIRSTFETCH 86  
     OPTIMIZERLEVEL 86  
     PARTITIONS 87  
     PAUSE 88  
     PLANONLY 88  
     PRESERVECONTEXT 93  
     PRINTLEVEL 94  
     read only 81  
     read repeatability 80  
     READONLY 94  
     READONLYDATABASE 95  
     RECOVERY 95  
     release lock 81  
     reserved word 3  
     RESTRICTION 96  
     RM 96  
     ROLLBACK 96  
     SCREEN 97  
     SCROLL 97  
     SCROLLROW 97  
     SPACE 98  
     SPANLIMIT 98  
     TIME 98  
     TIMEOUT 98  
     TIMEPICTURE 99  
     TIMESTAMP 99  
     TM 99  
     TRACE 99  
     TRACEFILE 100  
 SET CURSORNAME 16  
 SET DEFAULT STOGROUP 6  
 SET ERRORLEVEL 24  
 SET FILTER 3, 77  
 SET SCROLL ON 14, 16  
 SET SCROLLROW 14  
 SET SERVER 3, 101  
     DBDIR 102  
 SET SPOOL 4, 3, 102  
 Set trace  
     disable 99  
     enable 99  
 SHA 8  
 SHOW 4, 3, 104  
     CONNECT 105  
     LOG 105  
     REPORT 105  
     reserved word 3  
     ROWCOUNT 105

---

- SYSTEM 105
- VERSION 105
- SHOW CONNECTION 4, 3
  - command 106
- SHOW DATABASES 4, 3
  - LOCALLY 107
  - ON SERVER 107
- SHUTDOWN 3
- single quote
  - with bind variable 20
- SNAPSHOT
  - BACKUP 12
  - reserved word 3
  - RESTORE 55
- SORT 88
- SORTSPACE
  - reserved word 3
- source table
  - COPY 33
- SPACE
  - reserved word 3
- SET 98
- SPANLIMIT
  - reserved word 3
- SET 98
- SPOOL
  - reserved word 3
- SQL
  - command file 67
  - REPORT 67
  - reserved word 3
  - SAVE 67
  - SQL 67
- SQL command
  - parse 47
- SQL commands 5
- sql.ini 10
- SQLBASE
  - reserved word 3
- SQLGET 4, 109
  - reserved word 3
- SQLGSI
  - reserved word 3
- SQLSAB
  - reserved word 3
- SQLSCL
  - reserved word 3
- SQLSET 4, 111
  - reserved word 3
- SQLTalk
  - coding guidelines 10
  - command guidelines 10
  - entering commands 10
  - initialization file 11
  - purpose 2
  - reserved words 2
  - starting 7
- SQLTalk programs 6
- SS 23
- START AUDIT 6
- statement
  - tracing 99
- STOGROUP
  - reserved word 3
- STOP AUDIT 6
- STORE 4, 113
  - command name 114
  - procedure 114
  - procedure name 114
  - reserved word 3
  - SQL command 114
- stored command
  - ERASE 38
  - EXECUTE 40
  - owner 114
  - recompile 2, 4, 40
- stored commands
  - restriction mode 19, 57
- stored procedure
  - ERASE 38
  - EXECUTE 40
  - owner 114
- stored procedures 4
  - restriction mode 19, 57
- string 20
- string constants 10
  - specifying 10
- SUBTITLE
  - COLUMN 25
  - reserved word 3
- SUM
  - COMPUTE 26, 27
  - reserved word 3
- syntax
  - error
    - detect 47

---

SYSADM.SYSCOMMANDS 114

SYSTEM

reserved word 3  
SHOW 105

**T**

table

execution plan display 90  
row count 6  
TMPx 90

TCOUNT

reserved word 3

terminating connection 37

TIME

reserved word 3  
SET 98

TIMEOUT

reserved word 3  
SET 98

TIMEPICTURE

reserved word 3  
SET 99

TIMESTAMP

reserved word 3  
SET 99

tlk files 11

TM

reserved word 3  
SET 99

TMPx table 90

TO

BACKUP 13  
reserved word 3

TO BACKUP

ROLLFORWARD 62

TO END

ROLLFORWARD 61

TO TIME

ROLLFORWARD 62

TPONE

reserved word 3

TRACE

SET 99

trace

procedure 99  
output 100

TRACEFILE

SET 100

transactions

connection handles 71

TTITLE 3, 4, 115

DATE 116

PAGE 116

reserved word 3

**U**

UNDO 5, 16, 4, 117

reserved word 3

UNION 6, 19

UNLOAD 6

reserved word 3

with AUTORECOMPILE 4

UNLOCK DATABASE 6

UPDATE 6, 14

UPDATE STATISTICS 6

USE 3, 16, 4, 118

reserved word 3

user

name 10

username

how SQLBase determines 10

**V**

variables

bind data 47

verify

object 47

VERSION

reserved word 3

SHOW 105

**W**

WHERE CURRENT OF 15

WIDTH

COLUMN 20

reserved word 3

WKS

reserved word 3

WORK

reserved word 3

WRAP

COLUMN 25

**Y**

YY 23

YYYY 23