



*Open Tools from Sybase, Inc.*

**PowerBuilder**

**SQL Anywhere**

***User's Guide***

***Volume 2: Reference***

*Version 6*

# **Power Builder®**

AA0818

October 1997

Copyright © 1991-1997 Sybase, Inc. and its subsidiaries.

All rights reserved.

Printed in Ireland.

Information in this manual may change without notice and does not represent a commitment on the part of Sybase, Inc. and its subsidiaries.

The software described in this manual is provided by Powersoft Corporation under a Powersoft License agreement. The software may be used only in accordance with the terms of the agreement.

No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc. and its subsidiaries.

Sybase, Inc. and its subsidiaries claim copyright in this program and documentation as an unpublished work, revisions of which were first licensed on the date indicated in the foregoing notice. Claim of copyright does not imply waiver of other rights of Sybase, Inc. and its subsidiaries.

ClearConnect, Column Design, ComponentPack, InfoMaker, ObjectCycle, PowerBuilder, PowerDesigner, Powersoft, S-Designor, SQL SMART, and Sybase are registered trademarks of Sybase, Inc. and its subsidiaries. Adaptive Component Architecture, Adaptive Server Anywhere, Adaptive Server Enterprise, Adaptive Warehouse, AppModeler, DataArchitect, DataExpress, Data Pipeline, DataWindow, dbQueue, ImpactNow, InstaHelp, Jaguar CTS, jConnect for JDBC, MetaWorks, NetImpact, Optima++, Power++, PowerAMC, PowerBuilder Foundation Class Library, Power J, PowerScript, PowerSite, Powersoft Portfolio, Powersoft Professional, PowerTips, ProcessAnalyst, Runtime Kit for Unicode, SQL Anywhere, The Model For Client/Server Solutions, The Future Is Wide Open, Translation Toolkit, UNIBOM, Unilib, Uninull, Unisep, Unistring, Viewer, WarehouseArchitect, Watcom, Watcom SQL Server, Web.PB, and Web.SQL are trademarks of Sybase, Inc. or its subsidiaries. Certified PowerBuilder Developer and CPD are service marks of Sybase, Inc. or its subsidiaries. DataWindow is a patented proprietary technology of Sybase, Inc. or its subsidiaries.

AccuFonts is a trademark of AccuWare Business Solutions Ltd.

All other trademarks are the property of their respective owners.



## SQL Anywhere for PowerBuilder

This *SQL Anywhere User's Guide* describes all the features of the SQL Anywhere product. The version of SQL Anywhere that is included in PowerBuilder does not have all the features of the full SQL Anywhere product. The following features, documented in the *SQL Anywhere User's Guide* are not available to PowerBuilder users:

- ◆ **Embedded SQL, HLI, Open Client, and DDE interfaces**  
PowerBuilder users access SQL Anywhere through ODBC. The chapters on the SQL Anywhere programming interfaces can be ignored, and the Open Server Gateway to enable Open Client connections can also be ignored.
- ◆ **SQL Preprocessor** The SQL preprocessor is required only for Embedded SQL, and is not needed by or provided for PowerBuilder users.
- ◆ **SQL Remote** The SQL Remote replication technology is not included with PowerBuilder.
- ◆ **Multi-user network server** Only a standalone database engine is included with PowerBuilder. The multi-user network server is not included in PowerBuilder.
- ◆ **Operating systems** Only the version of SQL Anywhere for the operating system you are using is included in PowerBuilder. The *SQL Anywhere User's Guide* describes all platforms for which SQL Anywhere is available.

In addition, note that the sample database is referred to throughout by the file name SADEMO.DB. This sample database is the same as the Powersoft demo database (PSDEMODB.DB), and all examples in this book will work against the Powersoft demo database.



# Contents

<b>About This Manual .....</b>	<b>v</b>
--------------------------------	----------

## **PART ONE Introduction to SQL Anywhere**

<b>1</b>	<b>Introduction .....</b>	<b>3</b>
	About SQL Anywhere .....	4
	Upgrading databases to SQL Anywhere .....	7
	About this manual.....	8
<b>2</b>	<b>New Features in SQL Anywhere 5.....</b>	<b>13</b>
	What's in a name?.....	14
	New features overview .....	15
	New features in the Watcom-SQL language .....	18
	New sample database .....	21
<b>3</b>	<b>Overview of SQL Anywhere.....</b>	<b>23</b>
	The SQL Anywhere engine and the SQL Anywhere server .....	24
	Running SQL Anywhere on a single computer.....	25
	Running SQL Anywhere on a network .....	29
	Running mixed operating systems on a single computer.....	32
	SQL Anywhere programming interfaces .....	34
	The SQL Anywhere programs.....	36

## **PART TWO Tutorials**

<b>4</b>	<b>Managing Databases with Sybase Central .....</b>	<b>45</b>
	Sybase Central and database management .....	46
	Navigating the main Sybase Central window .....	47
	Adding a table to a database .....	52
	Viewing and editing procedures .....	57
	Managing users and groups.....	60

	Backing up a database using Sybase Central.....	63
	Using the Sybase Central online help.....	65
<b>5</b>	<b>Using ISQL.....</b>	<b>67</b>
	The SQL Anywhere program group.....	68
	Starting SQL Anywhere.....	69
	Connecting to the sample database from ISQL.....	70
	Accessing Help from ISQL.....	71
	The ISQL command window.....	72
	Leaving ISQL.....	73
	Displaying data in ISQL.....	74
	Command recall in ISQL.....	76
	Function keys.....	78
	Canceling an ISQL command.....	79
	What's next?.....	80
<b>6</b>	<b>Using Character-Mode ISQL.....</b>	<b>81</b>
	Tutorial files.....	82
	Starting the SQL Anywhere software.....	83
	Connecting to the sample database from ISQL.....	84
	ISQL menu selection.....	85
	Obtaining help from ISQL.....	86
	The ISQL command window.....	87
	Leaving ISQL.....	88
	Displaying data in ISQL.....	89
	Command window keys in ISQL.....	91
	Scrolling the data window.....	93
	Command recall in ISQL.....	94
	Function keys.....	95
	Aborting an ISQL command.....	96
	What next?.....	97
<b>7</b>	<b>Selecting Data from Database Tables.....</b>	<b>99</b>
	Looking at the information in a table.....	101
	Ordering query results.....	103
	Selecting columns from a table.....	104
	Selecting rows from a table.....	105
	Comparing dates in queries.....	106
	Compound search conditions in the WHERE clause.....	107
	Pattern matching in search conditions.....	108
	Matching rows by sound.....	109
	Short cuts for typing search conditions.....	110

<b>8</b>	<b>Joining Tables .....</b>	<b>111</b>
	Displaying a list of tables .....	112
	Joining tables with the cross product .....	114
	Restricting a join.....	115
	How tables are related.....	117
	Join operators.....	118
<b>9</b>	<b>Obtaining Aggregate Data .....</b>	<b>121</b>
	A first look at aggregate functions.....	122
	Using aggregate functions to obtain grouped data .....	123
	Restricting groups.....	125
<b>10</b>	<b>Updating the Database .....</b>	<b>127</b>
	Adding rows to a table .....	128
	Modifying rows in a table .....	129
	Canceling changes .....	130
	Making changes permanent.....	131
	Deleting rows.....	132
	Validity checking .....	133
<b>11</b>	<b>Introduction to Views .....</b>	<b>137</b>
	Defining a view.....	138
	Using views for security.....	140
<b>12</b>	<b>Introduction to Subqueries.....</b>	<b>141</b>
	Preparing to use subqueries .....	142
	A simple subquery.....	143
	Comparisons using subqueries .....	145
	Using subqueries instead of joins.....	148
<b>13</b>	<b>Command Files.....</b>	<b>151</b>
	Entering multiple statements in the ISQL Command window ..	152
	Saving statements as command files .....	153
	Command files with parameters .....	154
<b>14</b>	<b>System Tables .....</b>	<b>155</b>
	The SYSCATALOG table .....	156
	The SYSCOLUMNS table.....	157
	Other system tables.....	158

## PART THREE

## Using SQL Anywhere

<b>15</b>	<b>Connecting to a Database</b> .....	<b>161</b>
	Connection overview .....	162
	Connecting from the SQL Anywhere utilities .....	168
	Connecting from an ODBC-enabled application .....	169
<b>16</b>	<b>Designing Your Database</b> .....	<b>181</b>
	Relational database concepts .....	182
	Planning the database .....	185
	The design process .....	187
	Designing the database table properties.....	199
<b>17</b>	<b>Working with Database Objects</b> .....	<b>203</b>
	Using Sybase Central to work with database objects .....	204
	Using ISQL to work with database objects.....	205
	Working with databases .....	206
	Working with tables.....	211
	Working with views .....	217
	Working with indexes .....	223
<b>18</b>	<b>Ensuring Data Integrity</b> .....	<b>225</b>
	Data integrity overview .....	226
	Using column defaults.....	230
	Using table and column constraints.....	235
	Enforcing entity and referential integrity .....	239
	Integrity rules in the system tables .....	244
<b>19</b>	<b>Using Transactions and Locks</b> .....	<b>245</b>
	An overview of transactions .....	246
	How locking works .....	250
	Isolation levels and consistency .....	251
	How SQL Anywhere handles locking conflicts .....	254
	Choosing an isolation level .....	256
	Savepoints within transactions .....	258
	Particular concurrency issues.....	259
	Transactions and portable computers.....	262
<b>20</b>	<b>Using Procedures, Triggers, and Batches</b> .....	<b>265</b>
	Procedure and trigger overview .....	266
	Benefits of procedures and triggers.....	267

	Introduction to procedures .....	268
	Introduction to user-defined functions .....	273
	Introduction to triggers .....	276
	Introduction to batches .....	281
	Control statements .....	283
	The structure of procedures and triggers .....	286
	Returning results from procedures .....	290
	Using cursors in procedures and triggers .....	295
	Errors and warnings in procedures and triggers .....	300
	Using the EXECUTE IMMEDIATE statement in procedures....	307
	Transactions and savepoints in procedures and triggers .....	308
	Some hints for writing procedures.....	309
	Statements allowed in batches .....	312
	Calling external libraries from stored procedures .....	314
<b>21</b>	<b>Monitoring and Improving Performance.....</b>	<b>319</b>
	Factors affecting database performance.....	320
	Using keys to improve query performance.....	322
	Using indexes to improve query performance.....	326
	Search strategies for queries from more than one table.....	328
	Sorting query results.....	331
	Temporary tables used in query processing.....	332
	How the optimizer works.....	333
	Monitoring database performance .....	336
<b>22</b>	<b>Database Collations .....</b>	<b>347</b>
	Collation overview.....	348
	Support for multibyte character sets .....	352
	Choosing a character set.....	354
	Creating custom collations.....	357
	The collation file format.....	358
<b>23</b>	<b>Importing and Exporting Data .....</b>	<b>363</b>
	Import and export overview .....	364
	Exporting data from a database.....	366
	Importing data into a database .....	370
	Tuning bulk operations .....	373
<b>24</b>	<b>Managing User IDs and Permissions .....</b>	<b>375</b>
	An overview of database permissions.....	376
	Managing individual user IDs and permissions .....	380
	Managing groups.....	386
	Database object names and prefixes .....	391

	Using views and procedures for extra security .....	393
	How SQL Anywhere assesses user permissions.....	396
	Users and permissions in the system tables .....	397
<b>25</b>	<b>Backup and Data Recovery.....</b>	<b>399</b>
	System and media failures.....	400
	The SQL Anywhere logs .....	401
	Using a transaction log mirror .....	406
	Backing up your database .....	411
	Recovery from system failure .....	414
	Recovery from media failure.....	416
<b>26</b>	<b>Introduction to SQL Remote Replication.....</b>	<b>419</b>
	Introduction to data replication .....	420
	SQL Remote concepts .....	422
	SQL Remote features .....	427
	Message systems supported by SQL Remote .....	429
	Tutorial: setting up SQL Remote using Sybase Central .....	430
	Set up the consolidated database in Sybase Central .....	433
	Set up the remote database in Sybase Central.....	438
	Tutorial: setting up SQL Remote using ISQL and DBXTRACT .....	439
	Set up the consolidated database .....	442
	Set up the remote database .....	445
	Start replicating data .....	447
	A sample publication.....	450
	Some sample SQL Remote setups .....	451
<b>27</b>	<b>SQL Remote Administration.....</b>	<b>455</b>
	SQL Remote administration overview .....	456
	SQL Remote message types.....	457
	Managing SQL Remote permissions .....	463
	Setting up publications .....	470
	Designing publications .....	475
	Setting up subscriptions .....	483
	Synchronizing databases .....	484
	How statements are replicated by SQL Remote .....	489
	Managing a running SQL Remote setup: overview.....	494
	Running the SQL Remote Message Agent .....	496
	The SQL Remote message tracking system.....	500
	Transaction log and backup management for SQL Remote ....	503
	Error reporting and conflict resolution in SQL Remote.....	507
	Using passthrough mode for administration.....	513



<b>28</b>	<b>Running Programs as Services..... 517</b> Introduction to services..... 518 Managing services ..... 519 Adding and removing SQL Anywhere services ..... 520 Configuring SQL Anywhere services ..... 522 Starting and stopping services..... 526 Running more than one service ..... 528 Monitoring a SQL Anywhere network server service ..... 531 The Windows NT Control Panel Service Manager ..... 532
<b>PART FOUR</b>	
	<b>Transact-SQL Compatibility</b>
<b>29</b>	<b>Using Transact-SQL with SQL Anywhere..... 535</b> An overview of SQL Anywhere support for Transact-SQL..... 536 SQL Server and SQL Anywhere architectures ..... 539 General guidelines for writing portable SQL..... 543 Configuring SQL Anywhere for Transact-SQL compatibility .... 544 Using compatible data types..... 549 Local and global variables ..... 556 Building compatible expressions..... 561 Using compatible functions..... 565 Building compatible search conditions ..... 576 Other language elements ..... 580 Transact-SQL statement reference ..... 581 Compatible system catalog information ..... 598 SQL Server system and catalog procedures..... 601 Implicit data type conversion ..... 603
<b>30</b>	<b>Transact-SQL Procedure Language ..... 605</b> Transact-SQL procedure language overview ..... 606 Automatic translation of SQL statements ..... 608 Transact-SQL stored procedure overview..... 610 Transact-SQL trigger overview ..... 611 Transact-SQL batch overview ..... 613 Supported Transact-SQL procedure language statements ..... 614 Returning result sets from Transact-SQL procedures..... 624 Variable and cursor declarations..... 625 Error handling in Transact-SQL procedures ..... 627
<b>31</b>	<b>Using the Open Server Gateway ..... 629</b> Open Server Gateway overview ..... 630 Open Server Gateway architecture..... 632

Data type mappings .....	634
Setting up the Open Server Gateway .....	642
Events handled by Open Server Gateway .....	646
Using SQL Anywhere with OmniCONNECT .....	649

**PART FIVE**

**The SQL Anywhere Programming Interfaces**

<b>32</b>	<b>Programming Interfaces .....</b>	<b>653</b>
<b>33</b>	<b>The Embedded SQL Interface .....</b>	<b>655</b>
	The C language SQL preprocessor .....	656
	Embedded SQL interface data types .....	664
	Host variables .....	667
	The SQL communication area (SQLCA) .....	673
	Fetching data .....	677
	Static vs dynamic SQL .....	682
	The SQL descriptor area (SQLDA) .....	690
	SQL procedures in Embedded SQL .....	696
	Library functions .....	701
	Interface library DLL dynamic loading .....	723
	Embedded SQL commands .....	727
	Database examples .....	730
	SQLDEF.H header file .....	747
<b>34</b>	<b>ODBC Programming .....</b>	<b>751</b>
	ODBC C language programming .....	752
	ODBC programming for the Macintosh .....	762
<b>35</b>	<b>The WSQL DDE Server .....</b>	<b>765</b>
	DDE concepts .....	766
	Using WSQL DDE Server .....	768
	Excel and WSQL DDE Server .....	772
	Word and WSQL DDE Server .....	774
	Visual Basic and WSQL DDE Server .....	775
<b>36</b>	<b>The WSQL HLI Interface .....</b>	<b>779</b>
	DLL concepts .....	780
	Using WSQL HLI .....	781
	Host variables with WSQL HLI .....	782
	WSQL HLI functions .....	783
	wsqlexec command strings .....	790

WSQL HLI and Visual Basic.....	798
WSQL HLI and REXX.....	802

**PART SIX**

**SQL Anywhere Reference**

<b>37</b>	<b>SQL Anywhere Components .....</b>	<b>807</b>
	SQL Anywhere components overview .....	809
	Registry entries and environment variables .....	810
	Software component return codes .....	813
	The database engine .....	814
	The Backup utility.....	824
	The Collation utility.....	829
	The Compression utility.....	832
	The Erase utility .....	834
	The Information utility.....	837
	The Initialization utility.....	839
	The ISQL utility .....	846
	The Log Translation utility .....	849
	The Open Server Gateway .....	853
	The Open Server Information utility.....	855
	The Open Server Stop utility .....	856
	The REBUILD batch or command file.....	857
	The SQL Remote Database Extraction utility.....	858
	The SQL Remote Message Agent .....	863
	The Stop utility .....	867
	The Transaction Log utility .....	869
	The Uncompression utility .....	873
	The Unload utility .....	876
	The Upgrade utility .....	882
	The Validation utility .....	885
	The Write File utility .....	888
	The SQL Preprocessor .....	892
<b>38</b>	<b>Watcom-SQL Language Reference.....</b>	<b>895</b>
	Syntax conventions .....	896
	Watcom-SQL language elements.....	897
	Expressions.....	899
	Search conditions.....	907
	Comments in Watcom-SQL.....	915
<b>39</b>	<b>SQL Anywhere Data Types.....</b>	<b>917</b>
	Character data types .....	918

Numeric data types .....	920
Date and time data types .....	922
Binary data types .....	926
User-defined data types .....	927
Data type conversions .....	929
Year 2000 compliance .....	930

<b>40</b>	<b>Watcom-SQL Functions.....</b>	<b>935</b>
	Aggregate functions .....	936
	Numeric functions .....	938
	String functions .....	941
	Date and time functions .....	945
	Data type conversion functions .....	951
	System functions .....	953
	Miscellaneous functions .....	965

<b>41</b>	<b>Watcom-SQL Statements .....</b>	<b>969</b>
	ALLOCATE DESCRIPTOR statement.....	970
	ALTER DBSPACE statement.....	972
	ALTER PROCEDURE statement .....	974
	ALTER PUBLICATION statement .....	975
	ALTER REMOTE MESSAGE TYPE statement.....	976
	ALTER TABLE statement .....	977
	ALTER TRIGGER statement .....	982
	ALTER VIEW statement .....	983
	CALL statement .....	984
	CASE statement .....	986
	CHECKPOINT statement.....	988
	CLOSE statement.....	989
	COMMENT statement.....	991
	COMMIT statement .....	993
	Compound statements .....	995
	CONFIGURE statement.....	998
	CONNECT statement .....	999
	CREATE DATATYPE statement.....	1002
	CREATE DBSPACE statement.....	1004
	CREATE FUNCTION statement .....	1005
	CREATE INDEX statement.....	1007
	CREATE PROCEDURE statement .....	1009
	CREATE PUBLICATION statement .....	1013
	CREATE REMOTE MESSAGE TYPE statement.....	1015
	CREATE SCHEMA statement.....	1017
	CREATE SUBSCRIPTION statement .....	1019
	CREATE TABLE statement .....	1020

CREATE TRIGGER statement .....	1028
CREATE VARIABLE statement .....	1031
CREATE VIEW statement .....	1033
DBTOOL statement .....	1035
Declaration section .....	1038
DECLARE CURSOR statement .....	1039
DECLARE TEMPORARY TABLE statement .....	1043
DEALLOCATE DESCRIPTOR statement .....	1044
DELETE statement .....	1045
DELETE (positioned) statement .....	1047
DESCRIBE statement .....	1049
DISCONNECT statement .....	1052
DROP statement .....	1053
DROP CONNECTION statement .....	1055
DROP OPTIMIZER STATISTICS statement .....	1056
DROP PUBLICATION statement .....	1057
DROP REMOTE MESSAGE TYPE statement .....	1058
DROP STATEMENT statement .....	1059
DROP VARIABLE statement .....	1060
DROP SUBSCRIPTION statement .....	1061
EXECUTE statement .....	1062
EXECUTE IMMEDIATE statement .....	1064
EXIT statement .....	1065
EXPLAIN statement .....	1066
FETCH statement .....	1068
FOR statement .....	1073
FROM clause .....	1074
GET DATA statement .....	1081
GET DESCRIPTOR statement .....	1083
GET OPTION statement .....	1084
GRANT statement .....	1085
GRANT CONSOLIDATE statement .....	1089
GRANT PUBLISH statement .....	1091
GRANT REMOTE statement .....	1092
HELP statement .....	1094
IF statement .....	1095
INCLUDE statement .....	1097
INPUT statement .....	1098
INSERT statement .....	1102
LEAVE statement .....	1104
LOAD TABLE statement .....	1105
LOOP statement .....	1108
MESSAGE statement .....	1109
NULL value .....	1110
OPEN statement .....	1112
OUTPUT statement .....	1115

PARAMETERS statement.....	1118
PASSTHROUGH statement.....	1119
PREPARE statement.....	1120
PREPARE TO COMMIT statement.....	1123
PUT statement.....	1124
READ statement.....	1126
RELEASE SAVEPOINT statement.....	1127
RESIGNAL statement.....	1128
RESUME statement.....	1129
RETURN statement.....	1130
REVOKE statement.....	1132
REVOKE CONSOLIDATE statement.....	1134
REVOKE PUBLISH statement.....	1135
REVOKE REMOTE statement.....	1136
ROLLBACK statement.....	1137
ROLLBACK TO SAVEPOINT statement.....	1138
ROLLBACK TRIGGER statement.....	1139
SAVEPOINT statement.....	1140
SELECT statement.....	1141
SET statement.....	1145
SET CONNECTION statement.....	1147
SET DESCRIPTOR statement.....	1148
SET OPTION statement.....	1149
SET SQLCA statement.....	1170
SIGNAL statement.....	1171
START DATABASE statement.....	1172
START ENGINE statement.....	1173
START SUBSCRIPTION statement.....	1174
STOP DATABASE statement.....	1175
STOP ENGINE statement.....	1176
STOP SUBSCRIPTION statement.....	1177
SYNCHRONIZE SUBSCRIPTION statement.....	1178
SYSTEM statement.....	1179
TRUNCATE TABLE statement.....	1180
UNION operation.....	1181
UNLOAD TABLE statement.....	1182
UPDATE statement.....	1183
UPDATE (positioned) statement.....	1186
VALIDATE TABLE statement.....	1187
WHENEVER statement.....	1188

<b>SQL Anywhere Database Error Messages .....</b>	<b>1191</b>
Error message index by SQLCODE .....	1192
Error messages index by SQLSTATE .....	1201
Alphabetic list of error messages .....	1210

	Internal errors (assertion failed) .....	1296
<b>43</b>	<b>SQL Preprocessor Error Messages .....</b>	<b>1297</b>
	SQLPP errors .....	1298
	SQLPP warnings .....	1307
<b>44</b>	<b>Differences from Other SQL Dialects .....</b>	<b>1309</b>
	SQL Anywhere features.....	1310
<b>45</b>	<b>SQL Anywhere Limitations .....</b>	<b>1313</b>
	Size and number limitations .....	1314
<b>46</b>	<b>SQL Anywhere Keywords .....</b>	<b>1315</b>
	Alphabetical list of keywords.....	1316
<b>47</b>	<b>SQL Anywhere System Procedures and Functions.....</b>	<b>1319</b>
	System procedure overview .....	1320
	Catalog stored procedures.....	1321
	System extended stored procedures.....	1323
<b>48</b>	<b>SQL Anywhere System Tables .....</b>	<b>1329</b>
	System tables diagram.....	1330
	Alphabetical list of system tables.....	1331
<b>49</b>	<b>SQL Anywhere System Views .....</b>	<b>1355</b>
	Alphabetical list of views .....	1356
<b>50</b>	<b>Glossary.....</b>	<b>1363</b>





PART FIVE

# SQL Anywhere Reference


This part provides reference information for the Sybase SQL Anywhere database management system, including the software components, SQL language, error messages, system tables, and so on.



# SQL Anywhere Components

## About this chapter

SQL Anywhere includes a set of utility programs in addition to the database engine. The utility programs include tools for backing up databases and performing other database administration tasks. This chapter provides reference information for each of the SQL Anywhere components, including the database engine and the accompanying utility programs.

 Users of SQL Anywhere for Unix platforms, please see "SQL Anywhere components overview", on page 809.

## Contents

Topic	Page
SQL Anywhere components overview	809
Registry entries and environment variables	810
Software component return codes	813
The database engine	814
The Backup utility	824
The Collation utility	829
The Compression utility	832
The Erase utility	834
The Information utility	837
The Initialization utility	839
The ISQL utility	846
The Log Translation utility	849
The Open Server Gateway	853
The Open Server Information utility	855
The Open Server Stop utility	856
The REBUILD batch or command file	857
The SQL Remote Database Extraction utility	858
The SQL Remote Message Agent	863

<b>Topic</b>	<b>Page</b>
The Stop utility	867
The Transaction Log utility	869
The Uncompression utility	873
The Unload utility	876
The Upgrade utility	882
The Validation utility	885
The Write File utility	888
The SQL Preprocessor	892

## SQL Anywhere components overview

This chapter presents reference information on the programs and database administration utilities that are part of SQL Anywhere. The utilities can be accessed from Sybase Central, from ISQL, or as command-line programs.

For comprehensive documentation on Sybase Central, see the Sybase Central online Help. For an introduction to the Sybase Central database administration tool, see the chapter "Managing Databases with Sybase Central". For information on the SQL Anywhere Service Manager, see the chapter "Running Programs as Services".

The SQL Anywhere programs use a set of system environment variables. These variables are described in "Registry entries and environment variables" on page 810.

The SQL Anywhere programs also use a standard set of return codes. These return codes are described in "Software component return codes" on page 813.

### **SQL Anywhere for Unix platforms**

The syntax for starting SQL Anywhere components on Unix platforms differs slightly from other platforms. On Unix platforms such as Sun Solaris, IBM AIX, HP-UX or QNX, SQL Anywhere programs are executed with commands typed in the lower case, and program files do not have an .EXE suffix.

Versions of SQL Anywhere for Unix platforms have only the command line utilities, a character mode ISQL utility and a standalone version of the database available.

## Registry entries and environment variables

SQL Anywhere uses a set of system variables to store various types of information necessary for running the software. These system variables are stored in registries, INI files, or environment variables, depending on the operating system. Not all system variables need to be set in all circumstances.

The following is a list of system variables used by SQL Anywhere and a description of what they are used for.

### SQLANY environment variable

**Syntax**

**SQLANY** = *path*

**Description**

The SQLANY environment variable is used to contain the directory where SQL Anywhere is installed. The default installation directory is C:\SQLANY50. The install procedure automatically adds the SQLANY environment variable to your startup environment. The SQLANY variable is used by the batch files or command files that build the Embedded SQL examples.

In QNX, SQL Anywhere is installed in a fixed directory and the SQLANY variable is not required.

### SQLCONNECT environment variable

**Syntax**

**SQLCONNECT** = *keyword=value* ; ...

**SQLCONNECT** = *keyword#value* ; ...

**Description**

The SQLCONNECT environment variable specifies connection parameters that are used by several of the database tools to connect to a database engine or network server. This string is a list of parameter settings of the form **KEYWORD=value**, delimited by semicolons. The number sign "#" is an alternative to the equals sign, and should be used when setting the connection parameters string in the SQLCONNECT environment variable, as using "=" inside an environment variable setting is a syntax error.

The keywords are from the following table.

<b>Verbose keyword</b>	<b>Short form</b>
Userid	UID

Verbose keyword	Short form
Password	PWD
ConnectionName	CON
EngineName	ENG
DatabaseName	DBN
DatabaseFile	DBF
DatabaseSwitches	DBS
AutoStop	AutoStop
Start	Start
Unconditional	UNC
DataSourceName	DSN

☞ For a description of the connection parameters, see "Database connection parameters" in the chapter "Connecting to a Database".

## SQLPATH environment variable

<b>Syntax</b>	<b>SQLPATH</b> = <i>path</i> ;...
	<b>PATH</b> = <i>path</i> ;...
<b>Description</b>	ISQL searches along SQLPATH for ISQL command files and Help files before searching the system path.

## SQLREMOTE environment variable

<b>Syntax</b>	<b>SQLREMOTE</b> = <i>path</i>
<b>Description</b>	Addresses for the FILE message link in SQL Remote replication are subdirectories of the SQLREMOTE environment variable. This variable should point to a shared directory.

## SQLSTART environment variable

<b>Syntax</b>	<b>SQLSTART</b> = <i>start-line</i>
	<b>SQLSTARTW</b> = <i>start-line</i>

**Description** Historical. The SQLSTART environment variable information has been added to the SQLCONNECT environment variable as the start parameter.

## **TMP environment variable**

**Syntax**                    **TMP** = *directory*  
                              **TMPDIR** = *directory*  
                              **TEMP** = *directory*

**Description**                The database engine creates temporary files for various operations such as sorting and performing unions. These temporary files will be placed in the directory specified by the TMP, TMPDIR, or TEMP environment variables. (The database engine takes the first one of the three that it finds.)

If none of the environment variables is defined, temporary files are placed in the current directory.



## Software component return codes

All database components use the following executable return codes. The header file SQLDEF.H has constants defined for these codes.

<b>Code</b>	<b>Explanation</b>
0	Success
1	General failure
2	Invalid file format, and so on
3	File not found, unable to open, and so on
4	Out of memory
5	Terminated by user
6	Failed communications
7	Missing required database name
8	Client/server protocol mismatch
9	Unable to connect to database engine
10	Database engine not running
11	Database server not found
254	Reached stop time
255	Invalid parameters on command line

# The database engine

**Syntax** `dbeng50 [engine-switches][database-file [database-switches], ...]`  
`rtdsk50 [engine-switches][database-file [database-switches], ...]`

**Windows 3.x syntax** `dbeng50w [engine-switches][database-file [database-switches], ...]`  
`rtdsk50w [engine-switches][database-file [database-switches], ...]`  
`dbeng50s [engine-switches][database-file [database-switches], ...]`  
`rtdsk50s [engine-switches][database-file [database-switches], ...]`

Switch	Description
@filename	Read in switches from configuration file
@envvar	Read in switches from environment variable
-b	Run in bulk operations mode
-c cache-size	Set maximum cache size
-d	Disable asynchronous I/O (OS/2, Windows NT, NetWare only)
-df	Force direct I/O (Windows 3.1, DOS only)
-di	Use direct I/O if possible (Windows 3.1, DOS only)
-ga	Automatically shut down after last database closed
-gb level	Set database process priority class to level
-gc num	Set checkpoint timeout period
-gd level	Set database starting permission
-ge size	Sets the stack size for threads that run external functions
-gf	Disable firing of triggers
-gk level	Set permission for stopping the engine using DBSTOP
-gn num	Set number of threads
-gp size	Set maximum page size
-gr num	Set maximum recovery time
-gs size	Set thread stack size
-gw num	Set the interval (in milliseconds) for background processing
-gx	Disable dual threading
-m	Truncate transaction log after checkpoint
-n name	Name the database engine

Switch	Description
<b>-q</b>	Quiet mode—suppress output
<b>-ta <i>sec</i></b>	Scan time for terminated applications:—default 30 seconds
<b>-u</b>	Use buffered disk I/O (Windows 95 and Windows NT only)
<b>-v</b>	Log old values of all columns on UPDATE or DELETE for all databases
<b>-y</b>	Run as a Windows 95 service

**Recovery switches**

Switch	Description
<b>-a <i>log-file</i></b>	Apply named transaction log file
<b>-f</b>	Force database to start without transaction log

**Database switches**

Switch	Description
<b>-m</b>	Truncate transaction log after checkpoint
<b>-n <i>name</i></b>	Name the database
<b>-v</b>	Log old values of all columns on UPDATE or DELETE

**Description**

There are several versions of the SQL Anywhere database engine, and each version has a different executable name. The database engine for platforms other than Windows 3.x is named DBENG50.EXE, the 32-bit and 16-bit Windows 3.x executables being DBENG50W.EXE and DBENG50S.EXE respectively. The runtime database engine is named RTDSK50.EXE, with the 32-bit and 16-bit Windows 3.x executables being named RTDSK50W.EXE and RTDSK50S.EXE respectively.

The SQL Anywhere standalone database engine is completely compatible with the SQL Anywhere network server (DBSRV50.EXE). Client applications developed on one run against the other without alteration. For information about the network server and its command-line switches, see the *SQL Anywhere Network Guide*.

The 32-bit versions of the Windows 3.x database engine have much better performance than the 16-bit versions. They require Windows 3.x running in enhanced mode.

The database engine can be started with a number of database files or no database files. Command-line switches specified before any databases apply to the engine and all databases. Switches specified after a particular database file apply only to that database.

Client applications can load additional databases dynamically after the database engine has started. However, only databases of the same or smaller page size can be loaded dynamically.

If a *database-file* is specified without a file extension, SQL Anywhere first looks for *database-file* with extension WRT (a write file), followed by *database-file* with extension DB.

The database engines are programs that run as a separate task.

In DOS, the database engines are DOS **terminate and stay resident** programs (TSRs), meaning that they will load into the memory of your computer and then return control to DOS.

Runtime database engines are available with the SQL Anywhere Desktop Runtime System for DOS, Windows 3.x, OS/2 and Windows 95 or NT.

The runtime engine does not allow ALTER, CREATE, COMMENT or DROP commands. GRANT and REVOKE will allow you to add new users and change passwords but changing permissions on tables is not allowed. Also, the runtime database engine does not employ a transaction log (although it is a fully transaction-processing database engine) and does not support stored procedures and triggers.

**Supplying command-line switches in a configuration file**

A set of command line switches can be stored in a configuration file or in an environment variable. The database engine can be instructed to use these switches using the @ command-line switch.

**Engine switches**

**@filename** Read in command-line switches from the supplied file.

The file may contain line breaks, and may contain any set of command line switches. For example, the following command file holds a set of command line switches for an engine that starts with a cache size of 4 Mb, a name of myserver, and loads the sample database:

```
-c 4096  
-n myserver  
c:\sqlany50\sademo.db
```

If this configuration file is saved as C:\CONFIG.TXT, it can be used in an command line as follows:

```
DBENG50 @c:\config.txt
```

**@environment-variable** Read in command-line switches from the supplied environment variable. The environment variable may contain any set of command line switches. For example, the first of the following pair of statements sets an environment variable holding a set of command line switches for a database server that starts with a cache size of 4 Mb and loads the sample database. The second statement starts the database server:

```
set envvar=-c 4096 c:\sqlany50\sademo.db
DBENG50 @envvar
```

**-b** Use bulk operation mode. This is useful when loading large quantities of data into a database.

In bulk operations mode, the database server allows only one connection by one application. It does not keep a rollback log or a transaction log, and the multi-user locking mechanism is turned off. You should use a new log file when starting the database engine or server after loading data with the **-b** switch.

Bulk operation mode does not disable the firing of triggers.

☞ For more information on loading and unloading data, see "Tuning bulk operations" in the chapter "Importing and Exporting Data".

**-c cache-size** Set the size of the cache. The database server uses extra memory for caching database pages if it is set aside in the cache. Any cache size less than 10000 is assumed to be K-bytes (1K = 1024 bytes). Any cache size 10000 or greater is assumed to be in bytes. The cache size may also be specified as nK or nM (1M = 1024K). By default, the database server uses 2 megabytes of memory for caching. The more cache that can be given the engine, the better will be its performance.

**-d disable asynchronous I/O** This option applies to Windows NT, OS/2 and NetWare systems only. Use synchronous rather than asynchronous I/O. Asynchronous I/O is the default for these systems and should offer performance improvements.

**-df force direct I/O** This option applies to DOS and Windows 3.x systems only. Use direct I/O rather than DOS I/O. DOS I/O is the default. See the **-di** option for a description of direct I/O.

**-di use direct I/O if possible** This option applies for DOS and Windows 3.x only. Analyse the disk hardware and software and use direct I/O if possible. The default is to use DOS I/O.

As databases increase in size and get more fragmented, direct I/O can provide a significant increase in performance. However, direct I/O does not work with all disks. Using the `-di` option causes the engine or server to attempt to determine whether direct I/O will work with the current configuration. If direct I/O cannot be used, a message will be displayed on startup.

**Windows 95 I/O**

Asynchronous I/O is not supported in Windows 95 environments.

**-ga** Applications can cause databases to be started and stopped by the engine. Specifying this switch causes the engine to shutdown when the last database is stopped.

**-gb level** OS/2 and Windows NT only. Sets the database process priority class to *level*. Level must be one of **idle**, **normal** (the default), **high**, or **maximum**. **idle** is provided for completeness, and **maximum** may interfere with the running of your computer. **low** and **high** are the commonly used settings.

**-gc num** Sets the maximum desired length of time (in minutes) that the database server will run without doing a checkpoint.

When a database server is running with multiple databases, the checkpoint time specified by the first database started will be used unless overridden by this switch.

☞ For more information about checkpoints, see the `CHECKPOINT_TIME` option in "SET OPTION statement" in the chapter "Watcom-SQL Statements".

**-gd level** Set the database starting permission to *level*. This is the permission level required by a user to cause a new database file to be loaded by the server. The level can be one of the following:

- ◆ **dba** Only users with DBA authority can start new databases.
- ◆ **all** All users can start new databases (the default).
- ◆ **none** Starting new databases is not allowed.

**-ge size** Sets the stack size for threads running external functions, in bytes. The default is 16384 (16K). This switch is used only for OS/2, Windows NT, and NetWare.

**-gf** Disables firing of triggers by the server.

**-gk level** Sets the permission required to stop the database engine using DBSTOP to *level*. The level can be one of the following:

- ◆ **dba** Only users with DBA authority can use DBSTOP to stop the engine.
- ◆ **all** All users can use DBSTOP to stop the engine (the default).
- ◆ **none** The engine cannot be stopped using DBSTOP.

**-gn num** Sets the number of execution threads that will be used in the database server while running with multiple users.

☞ For more information, see the `THREAD_COUNT` option in "SET OPTION statement" in the chapter "Watcom-SQL Statements".

When a database server is running with multiple databases, the thread count specified by the first database started will be used unless overridden by this switch.

**-gp size** Sets the maximum page size allowed, in bytes. The size specified must be one of: 512, 1024, 2048, or 4096. When a database server is running with multiple databases, the page size specified by the first database will be used unless overridden by this switch. Without using this option, an attempt to load a database file with a page size larger than the page size of the database first loaded will fail.

**-gr num** Sets the maximum desired length of time (in minutes) that the database server will take to recover from system failure.

☞ For more information, see the `RECOVERY_TIME` option in "SET OPTION statement" in the chapter "Watcom-SQL Statements".

When a database server is running with multiple databases, the recovery time specified by the first database started will be used unless overridden by this switch.

**-gs size** Sets the stack size of every thread in the server. The value entered is multiplied by four to produce the stack size in bytes.

**-gw num** Sets the interval for background processing. At each interval, the engine carries out one I/O operation. The default setting is 500 (half a second).

**-gx** Disables dual threading. This option is available for the Windows 95, Windows NT, OS/2, and NetWare versions.

**-m** Truncate (delete) the transaction log when a checkpoint is done, either at shutdown or as a result of a checkpoint scheduled by the database engine. This provides a way to automatically limit the growth of the transaction log. Checkpoint frequency is still controlled by the CHECKPOINT\_TIME and RECOVERY\_TIME options (also definable on the command line).

The -m option is useful where high volume transactions requiring fast response times are being processed, and the contents of the transaction log are not being relied upon for recovery or replication. When this option is selected, there is no protection provided against media failure on the device containing the database files.

To avoid database file fragmentation, it is recommended that where this option is used, the transaction log be placed on a separate device or partition from the database itself.

**Replicated databases**

Do not use the -m option with databases that are being replicated as replication inherently relies on transaction log information.

**-n name** Set the name of the database server. By default, the database server receives the name of the database file with the path and extension removed. For example, if the server is started on C:\SQLANY50\SADEMO.DB and no -n switch is specified, then the name of the server will be sademo.

The server name can be used on the connect statement to specify to which server you wish to connect. In all environments, there is always a default database server that will be used if no server name is specified provided at least one database server or SQL Anywhere Client (DBCLIENT) is running on the computer.

**-q** Operate quietly. Suppress all output.

**-ta seconds** For Windows 3.x, Windows NT, and Windows 95 only. The database engine periodically scans the connection list and disconnects any connections associated with terminated applications. The scan period can be controlled using the -ta switch, and has a default value of 30 seconds. Setting the value to zero prevents scanning.

**-u** Files are opened using the operating system disk cache in addition to the database cache. This option applies to the Windows 95 and Windows NT database servers only. While the operating system disk cache may improve performance in some cases, in general better performance is obtained without this switch, using the database cache only.



**-v** Cause the database engine to record in the appropriate transaction log the previous values of each of the columns whenever a table row is updated or deleted. By default, the engine will only record enough information to uniquely identify the row (primary key values or values from a not null unique index). This switch is useful for working on a copy of a database file.

This option does not apply to the runtime database engine, which does not support transaction logs, or database files that are not using a transaction log.

**-y** Runs database engine as a Windows 95 service. By registering the database engine as a Windows 95 service it continues to operate whether as users log on or off and shutdown commands are ignored.

☞ The database engine can also be run as an NT service. For more information, see the chapter "Running Programs as Services".

## Recovery switches

**-a log-file** Apply the named transaction log. This is used to recover from media failure on the database file. When this option is specified, the database server will apply the log and then terminate—it will not continue to run.

This option does not apply to the runtime database engine, which does not support transaction logs, or database files that are not using a transaction log.

☞ For more information on recovery from media failure, see the chapter "Backup and Data Recovery".

**-f** This option is used for recovery: either to force the database server to start after the transaction log has been lost, or to force the database server to start using a transaction log it would otherwise not find.

If there is no transaction log, the database server carries out a checkpoint recovery of the database and then terminates—it does not continue to run. You can then restart the database server without the -f option for normal operation.

If there is a transaction log in the current directory, the database server carries out a checkpoint recovery, and a recovery using the transaction log, and then terminates—it does not continue to run. You can then restart the database server without the -f option for normal operation.

This option does not apply to the runtime database engine, which does not support transaction logs.

☞ For more information on recovery, see the chapter "Backup and Data Recovery".

### Database switches

**-m** Truncate (delete) the transaction log when a checkpoint is done, either at shutdown or as a result of a checkpoint scheduled by the engine. This provides a way to automatically limit the growth of the transaction log. Checkpoint frequency is still controlled by the CHECKPOINT\_TIME and RECOVERY\_TIME options (also definable on the command line).

The -m option is useful where high volume transactions requiring fast response times are being processed, and the contents of the transaction log are not being relied upon for recovery or replication. When this option is selected, there is no protection provided against media failure on the device containing the database files.

To avoid database file fragmentation, it is recommended that where this option is used, the transaction log be placed on a separate device or partition from the database itself.

This switch is the same as the -m engine switch, but applies only to the database identified by the *database-file* command-line variable.

#### Replicated databases

Do not use the -m option with databases that are being replicated as replication inherently relies on transaction log information.

**-n name** Set the name of the database. Both database servers and databases can be named. Since a database server can load several databases, the database name is used to distinguish the different databases.

By default, the database receives the name of the file with the path and extension removed. For example, if the server is started on C:\SQLANY50\SADEMO.DB and no -n switch is specified, then the name of the database is sademo.

**-v** Causes the database engine to record in the transaction log the previous values of each of the columns whenever a row of the specified database is updated or deleted. By default, the engine will only record enough information to uniquely identify the row (primary key values or values from a not null unique index). This switch is useful for working on a copy of a database file.

This option does not apply to the runtime database engine, which does not support transaction logs, or database files that are not using a transaction log.

The backup, initialization, unload, and validation utilities automatically load the database engine if it has not been previously loaded. The SQL Anywhere Client (DBCLIENT) can also be started automatically in this way. See "Registry entries and environment variables" on page 810 and the respective commands for more details. If the database engine (or client) is started automatically, it will be unloaded automatically when the software is finished executing.

## The Backup utility


With the Backup utility, you can back up running databases, database files, transaction logs, and write files.

You can access the Backup utility in the following ways:

- ◆ From Sybase Central, for interactive use under Windows 95 or NT.
- ◆ From the ISQL Database Tools window.
- ◆ From the ISQL DBTOOL statement.
- ◆ From the system command line, using the DBBACKUP utility. This is useful for incorporating backup procedures into batch or command files.

The Backup utility makes a backup copy of all the files making up a single database. A simple database consists of two files: the main database file and the transaction log. More complicated databases can store tables in multiple files. Each file is a separate dbspace. All backup filenames are the same as the database filenames.

Running the Backup utility on a running database is equivalent to copying the database files when the database is not running. It is provided to allow a database to be backed up while other applications or users are using the database.

 For more information

For a description of suggested backup procedures, see the chapter "Backup and Data Recovery".

## Backing up a database from Sybase Central

### ❖ To back up a running database:

- 1 Connect to the database.
- 2 Right-click the database and click Backup in the popup menu. The Backup Wizard is displayed.
- 3 Follow the instructions in the Wizard.

### ❖ To back up a database file or a running database:

- 1 Open the Database Utilities folder in the left panel.
- 2 Double-click Backup Database in the right panel.

- 3 Follow the instructions in the wizard.

☞ For full information on backing up a database from Sybase Central, see the Sybase Central online Help. For more information about options, see "Backup utility options" on page 826.

## Backing up a database from the ISQL Database Tools window

### ❖ To use the Backup utility from the ISQL Database Tools window:

- 1 Select Database Tools from the Window menu.
- 2 Click Backup Database Files on the Tools list.
- 3 Enter a user ID and password to use when connecting to the database.
- 4 For a running database, enter a database name and server name (if more than one is running). For a database file, enter the filename (with path) and optionally a start line to specify command-line switches for the database engine or SQL Anywhere Client.
- 5 Click Backup, and select from the options displayed in the dialog.
- 6 Click OK to back up the database.

## Backing up a database using the DBTOOL statement

### Syntax

The syntax to access the Backup utility from the ISQL DBTOOL statement is as follows:

```
DBTOOL BACKUP TO directory
... [DBFILE] [ WRITE FILE ] [ [ TRANSACTION ] LOG ]
    | [ ALL FILES ]
... | [ RENAME [ TRANSACTION ] LOG ]
    | [ TRUNCATE [ TRANSACTION ] LOG ]
... [ NOCONFIRM ] USING connection-string
```

### Example

The following statement connects to and backs up the sample database, to directory C:\TEMP.

```
DBTOOL BACKUP TO 'c:\temp' DBFILE
USING 'dbf=c:\sqlany50\sademo.db;uid=dba;pwd=sql'
```


## The DBBACKUP command-line utility

**Syntax** `dbbackup [switches] directory`

**Windows 3.x syntax** `dbbackw [switches] directory`

Switch	Description
-c "keyword=value; ..."	Supply database connection parameters
-d	Only back up main database file
-k	Change backup transaction log naming convention
-l file	Live backup of transaction log to file
-o file	Log output messages to file
-q	Quiet mode—do not print messages
-r	Rename and start new transaction log
-t	Only back up transaction log
-w	Only back up write file
-x	Delete and restart transaction log
-y	Replace files without confirmation

If none of the switches -d, -t, or -w are used, all database files are backed up.

 For more information about the command-line switches, see "Backup utility options", next.

## Backup utility options

**Connection parameters** For a description of the connection parameters, see "Database connection parameters" in the chapter "Connecting to a Database". If the connection parameters are not specified, connection parameters from the SQLCONNECT environment variable are used, if set. The user ID must have DBA authority.

For the DBBACKUP command-line utility, this is the -c command-line switch. For example, the following statement backs up the sademo database running on the server sample\_server, connecting as user ID DBA with password SQL:

```
dbbackup -c  
"eng=sample_server;dbn=sademo;uid=dba;pwd=sql"
```

**Backup main database only** Back up the main database files only, without backing up the transaction log file or a write file, if one exists. For the DBBACKUP command-line utility, this is the `-d` command-line switch.

**Change backup transaction log naming convention** This option changes the naming convention of the backup transaction log file to `YYMMDDxx.LOG`, where `xx` is a number from 00 to 99 and `YYMMDD` represents the current year, month and day. By default the name used for the backup transaction log file is identical to the file name of the transaction log being backed up. For the DBBACKUP command-line utility, this is the `-k` command-line switch.

**Live backup** This option is provided to enable a secondary system to be brought up rapidly in the event of a server crash. A live backup does not terminate, but continues running while the server runs. It runs until the primary server crashes. At that point it is shut down, but the backed up log file is intact and can be used to bring a secondary system up quickly. For the DBBACKUP command-line utility, this is the `-l` (lower case L) command-line switch.

**Output messages to log** Sends backup messages to a named log file. For the DBBACKUP command-line utility, this is the `-o` command-line switch.

**Operate quietly** Do not display messages on a window. For the DBBACKUP command-line utility, this is the `-q` command-line switch. This option is not available from other environments.

**Rename and start new transaction log** This option forces a checkpoint and the following three steps to occur:

- ◆ **Step 1** A copy is made of the current working transaction log file and saved to the directory specified in the command line.
- ◆ **Step 2** The current transaction log remains in its current directory, but is renamed using the format `YYMMDDxx.LOG`, where `xx` is a number from 00 to 99 and `YYMMDD` represents the current year, month and day. This file is then no longer the current transaction log.
- ◆ **Step 3** A new transaction log file is generated that contains no transactions. It is given the name of the file previously considered the current transaction log and is used by the database engine as the current transaction log.

For the DBBACKUP command-line utility, this is the `-r` command-line switch.

**Back up the transaction log file only** This can be used as an incremental backup since the transaction log can be applied to the most recently backed up copy of the database file(s). For the DBBACKUP command-line utility, this is the `-t` command-line switch.

**Back up the database write file only** See DBWRITE for a description of database write files. For the DBBACKUP command-line utility, this is the `-w` command-line switch.

**Delete and restart the transaction log** With this option, the existing transaction log is backed up, then the original is deleted and a new transaction log is started with the same name. This option causes the backup to wait for a point when all transactions from all connections are committed. For the DBBACKUP command-line utility, this is the `-x` command-line switch.

**Operate without confirming actions** Without this option, you are prompted to confirm the creation of the backup directory or the replacement of a previous backup file in the directory. For the DBBACKUP command-line utility, this is the `-y` command-line switch.



## The Collation utility

With the Collation utility, you can extract a collation (sorting sequence) from the SYS.SYSCOLLATION system table of a database into a file suitable for creating a database using a custom collation.

The file produced by the Collation utility can be modified and used with Sybase Central or the `-z` option of DBINIT to create a new database with a custom collation.

Ensure that the label is changed on the line that looks like:

```
Collation label (name)
```

Otherwise, the collation cannot be used to create a database.

If you wish to create a custom collation but have not yet created a database, you should extract a collation from the sample database provided with SQL Anywhere.

☞ For more information on custom collating sequences, see the chapter "Database Collations".

You can access the Collation utility in the following ways:

- ◆ From the ISQL Database Tools window.
- ◆ From the ISQL DBTOOL statement.
- ◆ From the system command line, using the DBCOLLAT command-line utility.

### Extracting a collation in the ISQL Database Tools window

❖ **To use the Collation utility from the ISQL Database Tools window:**

- 1 Select Database Tools from the Window menu.
- 2 Click Extract Collation from Database on the Tools list.
- 3 For a running database, enter a database name and server name (if more than one is running). For a database file, enter the filename (with path) and optionally a start line to specify command-line switches for the database engine or SQL Anywhere Client.
- 4 Click Extract, and select from the options displayed in the dialog.
- 5 Click OK to extract the collation file from the database.

## Extracting a collation using the DBTOOL statement

**Syntax** The syntax to access the Collation utility from the ISQL DBTOOL statement is as follows:

```
DBTOOL UNLOAD COLLATION [ name ] TO filename
... USING connection-string
... [ EMPTY MAPPINGS ] [ HEX | HEXADEDECIMAL ] [ NOCONFIRM ]
```

## The DBCOLLAT command-line utility

**Syntax** `dbcollat [switches] output-file`

**Windows 3.x syntax** `dbcollw [switches] output-file`

Switch	Description
<code>-c "keyword=value; ..."</code>	Supply database connection parameters
<code>-e</code>	Include empty mappings
<code>-o filename</code>	Output log messages to file
<code>-q</code>	Quiet mode — do not print messages
<code>-x</code>	Use hex for extended characters (7F-FF)
<code>-y</code>	Replace file without confirmation
<code>-z col-seq</code>	Specify collating sequence label

☞ For more information about the command-line switches, see "Collation utility options", next.

## Collation utility options

**Connection parameters** For a description of the connection parameters, see "Database connection parameters" in the chapter "Connecting to a Database". If the connection parameters are not specified, connection parameters from the SQLCONNECT environment variable are used, if set. The user ID must have DBA authority.

For the DBCOLLAT command-line utility, this is the `-c` command-line switch. For example, the following statement extracts a collation file from the sademo database running on the sample\_server server, connecting as user ID DBA with password SQL:

```
dbcollat -c
"eng=sample_server;dbn=sademo;uid=dba;pwd=sql "
c:\sample\col
```

**Include empty mappings** Normally, collations don't specify the actual value that a character is to sort to. Instead, each line of the collation sorts to one position higher than the previous line. However, older collations have gaps between some sort positions. Normally, the Collation utility skips the gaps and writes the next line with an explicit sort-position. This option causes the Collation utility to write empty mappings (consisting of just a colon (:)) for each line in the gap. For the DBCOLLAT command-line utility, this is the `-e` command-line switch.

**Output log messages to file** Redirect the log messages from the Collation utility to a named file. For the DBCOLLAT command-line utility, this is the `-o` command-line switch.

**Operate quietly** Do not display messages on a window. For the DBCOLLAT command-line utility, this is the `-q` command-line switch. This option is not available from other environments.

**Use hexadecimal for extended characters (7F to FF)** Extended single-byte characters (whose value is greater than hex 7F) may or may not appear correctly on your screen, depending on whether or not the code page in use on your computer is the same as the code page being used in the collation you are extracting. This option causes the Collation utility to write all characters at hex 7F or above as a two-digit hexadecimal number, in the form:

```
\xdd
```

(For example, `\x80`, `\xFE`). Normally, only characters from hex 00 to hex 1F, hex 7F and hex FF are written in hexadecimal form. For the DBCOLLAT command-line utility, this is the `-x` command-line switch.

**Operate without confirming actions** Without this option, you are prompted to confirm replacing an existing collation file. For the DBCOLLAT command-line utility, this is the `-y` command-line switch.

**Specify collating sequence label** Specify the label of the collation to be extracted. The names of the collation sequences can be found in the `collation_label` column of the `SYS.SYSCOLLATION` table. If this option is not specified, then the Collation utility extracts the collation being used by the database. For the DBCOLLAT command-line utility, this is the `-z` command-line switch.

## The Compression utility

With the Compression utility you can compress a database file. The Compression utility reads the given database file and create a compressed database file. Compressed database files are useful if disk space is limited. Compressed databases are usually 40 to 60 per cent of their original size. The database engine cannot update compressed database files: they must be used in conjunction with write files.

The Compression utility does not compress files other than the main database file.

You can access the Compression utility in the following ways:

- ◆ From Sybase Central, for interactive use under Windows 95 or NT.
- ◆ From the ISQL Database Tools window.
- ◆ From the ISQL DBTOOL statement.
- ◆ From the system command line, using the DBSHRINK command-line utility. This is useful for incorporating into batch or command files.

### Compressing a database in Sybase Central

❖ **To compress a database file:**

- 1 Open the Database Utilities folder in the left panel.
- 2 Double-click Compress Database in the right panel. The Compress Database Wizard is displayed.
- 3 Follow the instructions in the Wizard.

☞ For full information on compressing a database in Sybase Central, see the Sybase Central online Help.

### Compressing a database from the ISQL Database Tools window

❖ **To use the Compression utility from the ISQL Database Tools window:**

- 1 Select Database Tools from the Window menu.
- 2 Click Compress Database on the Tools list.

- 3 Enter the database filename (with path).
- 4 Click Compress, and enter a filename to use for the uncompressed database.
- 5 Click OK to compress the selected database file.

## Compressing a database using the DBTOOL statement

### Syntax

The syntax for the Compression utility from the ISQL DBTOOL statement is as follows:

```
DBTOOL COMPRESS DATABASE filename TO filename ]
```

## The DBSHRINK command-line utility

### Syntax

```
dbshrink [switches] database-file [compressed-database-file]
```

### Windows 3.x syntax

```
dbshrinw [switches] database-file [compressed-database-file]
```

Switch	Description
-q	Quiet mode—do not print messages
-y	Erase existing output file without confirmation

DBSHRINK reads the given *database* file and create a compressed database file. The compressed filename defaults to the same name as the first with an extension of CDB. The output filename (with extension) must not have the same name as the input filename (with extension).

☞ For more information about the command-line switches, see "Compression utility options", next.

## Compression utility options

**Operate quietly** Do not display messages on a window. For the DBSHRINK command-line utility, this is the -q command-line switch. This option is not available from other environments.

**Operate without confirming actions** Without this option, you are prompted to confirm the replacement of an existing database file. For the DBSHRINK command-line utility, this is the -y command-line switch.

## The Erase utility

With the Erase utility, you can erase a database file or write file and its associated transaction log, or you can erase a transaction log file or transaction log mirror file. All SQL Anywhere database files, write files, and transaction log files are marked read-only to prevent accidental damage to the database or accidental deletion of the database files. Deletion of a database file that references other dbspaces does not automatically delete the dbspace files.

If you erase a database file or write file, the associated transaction log and transaction log mirror are also deleted. If you erase a transaction log for a database that also maintains a transaction log mirror, the mirror is not deleted.

You can access the Erase utility in the following ways:

- ◆ From Sybase Central, for interactive use under Windows 95 or NT.
- ◆ From the ISQL Database Tools window.
- ◆ From the ISQL DBTOOL statement.
- ◆ From the system command line, using the DBERASE command-line utility. This is useful for incorporating into batch or command files.

### Erasing a database from Sybase Central

❖ **To erase a database file:**

- 1 Open the Database Utilities folder in the left panel.
- 2 Double-click Erase Database in the right panel. The Erase a Database Wizard is displayed.
- 3 Follow the instructions in the Wizard.

🔗 For full information on erasing a database from Sybase Central, see the Sybase Central online Help.

### Erasing a database from the ISQL Database Tools window

❖ **To use the Erase utility from the ISQL Database Tools window:**

- 1 Select Database Tools from the Window menu.

- 2 Click Erase Database or Write File on the Tools list.
- 3 Enter the filename (with path).
- 4 Click Erase, and select from the options displayed in the dialog.
- 5 Click OK to erase the selected files.

## Erasing a database using the DBTOOL statement

### Syntax

The syntax for the Erase utility from the ISQL DBTOOL statement is as follows:

```
drop database database-file [ noconfirm ]
```

The *database-file* may be a database file, write file, or transaction log file. The full filename must be specified, including extension. If a database file or write file is specified, the associated transaction log file (and mirror, if one is maintained) is also erased.

### Example

The following statement erases the database file C:\TEMP.DB and its associated transaction log file, prompting to confirm deletion.

```
DBTOOL DROP DATABASE 'c:\temp.db'
```

## The DBERASE command-line utility

### Syntax

```
dberase [switches] database-file
```

### Windows 3.x syntax

```
dberasew [switches] database-file
```

Switch	Description
-o	Output log messages to file
-q	Operate quietly—do not print messages
-y	Erase files without confirmation

The *database-file* may be a database file, write file, or transaction log file. The full filename must be specified, including extension. If a database file or write file is specified, the associated transaction log file (and mirror, if one is maintained) is also erased.

☞ For more information about the command-line switches, see "Erase utility options", next.

## Erase utility options

**Output log messages to file** Redirect log messages to the named file. For the DBERASE command-line utility, this is the -o option.

**Operate quietly** Do not display messages on a window For the DBERASE command-line utility, this is the -q command-line switch. This option is not available from other environments.

**Operate without confirming actions** Without this option, you are prompted to confirm the deletion of each file. For the DBERASE command-line utility, this is the -y command-line switch.



## The Information utility

With the Information utility you can display information about a database file or write file. The database you wish to examine should not be running when you run the Information utility. The utility indicates when the database was created, the name of any transaction log file or log mirror that is maintained, the page size and other information. Optionally, it can also provide table usage statistics and details.

You can access the Information utility in the following ways:

- ◆ From the ISQL Database Tools window.
- ◆ From the ISQL DBTOOL statement.
- ◆ From the system command line, using the DBINFO command-line utility.

### Obtaining database information in the ISQL Database Tools window

#### ❖ To use the Information utility from the ISQL Database Tools window:

- 1 Select Database Tools from the Window menu.
- 2 Click Database Information on the Tools list.
- 3 Enter the database filename (with path).
- 4 Click Display to show information about the database.
- 5 Click Page Usage to show information on database pages.

### Obtaining database information using the DBTOOL statement

#### Syntax

The syntax for the Information utility from the ISQL DBTOOL statement is as follows:

```
DBTOOL DBINFO DATABASE database-file TO output-file
...      [ [ WITH ] PAGE USAGE ]
...      USING connection-string
```

A database or write filename, with path, must be supplied. The database you wish to examine should not be running when you run the Information utility.

**Example**

The following statement gets information about the sample database and puts it in the file C:\TEMP.TXT.

```
DBTOOL DBINFO DATABASE 'c:\sqlany50\sademo.db'  
TO 'c:\temp.txt'  
WITH PAGE USAGE  
USING 'uid=dba;pwd=sql'
```

## The DBINFO command-line utility

**Syntax**

**dbinfo** [switches] filename

**Windows 3.x  
syntax**

**dbinfo** [switches] filename

Switch	Description
-o filename	Output messages to file
-q	Suppress any messages
-u	Output page usage statistics

☞ For more information about the command-line switches, see "Information utility options", next.

## Information utility options

**Database file name** A database or write file name, with path, must be supplied. The database you wish to examine should not be running when you run the Information utility.

**Output log messages to file** Redirect log messages to the named file. For the DBINFO command-line utility, this is the -o command-line switch.

**Operate quietly** Do not display the information. The return code may still provide useful information (see "Software component return codes" on page 813). For the DBINFO command-line utility, this is the -q command-line switch. This option is only available from the command-line utility.

**Page usage statistics** Information about the usage and size of all tables, including system and user-defined tables, is available as an option by using the -u command-line switch.

## The Initialization utility

With the Initialization utility you can initialize (create) a database. A number of database attributes are specified at initialization and cannot be changed later except by unloading, reinitializing, and rebuilding the entire database:

- ◆ Case sensitivity or insensitivity
- ◆ Storage as an encrypted file
- ◆ Treatment of trailing blanks in comparisons
- ◆ The page size
- ◆ The collation sequence used

In addition, the choice of whether to use a transaction log and a transaction log mirror is made at initialization. This choice can be changed later using the transaction log utility.

You can access the Initialization utility in the following ways:


- ◆ From Sybase Central, for interactive use under Windows 95 or NT.
- ◆ From the ISQL Database Tools window.
- ◆ From the ISQL DBTOOL statement.
- ◆ From the system command line, using the DBINIT command-line utility. This is useful for incorporating into batch or command files.

The standalone database engine is required by the initialization utility during the initialization process.

## Creating a database in Sybase Central

### ❖ To create a database:

- 1 Open the Database Utilities folder in the left panel.
- 2 Double-click Create Database in the right panel. The Database Creation Wizard is displayed.
- 3 Follow the instructions in the Wizard.

 For full information on creating a database in Sybase Central, see the Sybase Central online Help.

## Creating a database from the ISQL Database Tools window

❖ **To use the Initialization utility from the ISQL Database Tools window:**

- 1 Select Database Tools from the Window menu.
- 2 Click Create Database on the Tools list.
- 3 Enter the database filename (with path).
- 4 Click Create, and select from the options displayed in the dialog.
- 5 Click OK to create the database.

## Creating a database using the DBTOOL statement from within ISQL

**Syntax**

The syntax for the Initialization utility from the ISQL DBTOOL statement is as follows:

```
DBTOOL CREATE DATABASE filename
...      | [ NO [ TRANSACTION ] LOG ]      |
          | [ [ TRANSACTION ] LOG TO filename ] |
...      | [ IGNORE CASE ]                    |
          | [ RESPECT CASE ]                |
...      | [ PAGE SIZE n ] [ COLLATION name ] |
...      | [ ENCRYPT ] [ TRAILING SPACES ]     |
```

## The DBINIT command-line utility

**Syntax**

```
dbinit [switches] new-database-file
```

**Windows 3.x  
syntax**

```
dbinitw [switches] new-database-file
```

Switch	Description
-b	Blank padding of strings for comparisons
-c	Case sensitivity for all string comparisons
-e	Encrypt database
-g <i>user</i>	User name as replacement for dbo
-k	Omit Watcom SQL compatibility views SYS.SYSCOLUMNS and SYS.SYSINDEXES

Switch	Description
<b>-l</b>	List available collating sequences
<b>-m</b> <i>file-name</i>	Use a transaction log mirror (default is no mirror)
<b>-n</b>	No transaction log
<b>-o</b> <i>filename</i>	Output messages to file
<b>-p</b> <i>page-size</i>	Set page size
<b>-q</b>	Quiet mode—do not print messages
<b>-t</b> <i>log-name</i>	Transaction log filename (default is database name with LOG extension)
<b>-z</b> <i>col-seq</i>	Collation sequence used for comparisons

For example, the database TEST.DB can be created with 1024 byte pages as follows:

```
dbinit -p 1024 test.db
```

☞ For more information about the command-line switches, see "Initialization utility options", next.

## Initialization utility options

**Ignore trailing blanks** Trailing blanks are ignored for comparison purposes, and embedded SQL programs pad strings fetched into character arrays. For example, the two strings

```
'Smith'
'Smith   '
```

would be treated as equal in a database created with trailing blanks ignored.

This option is provided for compatibility with the ISO/ANSI SQL standard, which is to ignore trailing blanks in comparisons. The default is that blanks are significant for comparisons, which was the only behavior supported in releases up to Watcom SQL 3.0. For the DBINIT command-line utility, this is the **-b** command-line switch.

**Case sensitivity for all string comparisons** For databases created with this option, all values are considered to be case sensitive in comparisons and string operations.

This option is provided for compatibility with the ISO/ANSI SQL standard. The default is that all comparisons are case insensitive, which was the only behavior supported up to release Watcom SQL 3.0. For the DBINIT command-line utility, this is the `-c` command-line switch.

**User ID and password**

All databases are created with at least one user ID, DBA, with password SQL. If you specify create a database requiring case-sensitive comparisons, the DBA user ID and its password must be entered in uppercase.

**Encrypt database** Encryption makes it more difficult for someone to decipher the data in your database by using a disk utility to look at the file. File compaction utilities are not able to compress encrypted database files as much as unencrypted ones. For the DBINIT command-line utility, this is the `-e` command-line switch.

**User ID for dbo user** SQL Anywhere contains a set of system views that mimic the system tables of Sybase SQL Server. By default, the owner of these views is the user ID `dbo`, which is the same as the owner of the SQL Server system tables. If you already have a user ID named `dbo`, or you wish to use that user ID for other purposes, this option allows you to provide an alternative user ID for the owner of the SQL Server-like system views. For the DBINIT command-line utility, this is the `-g` command-line switch.

**Omit Watcom SQL compatibility views** By default, the DBINIT command-line utility generates the views `SYS.SYSCOLUMNS` and `SYS.SYSINDEXES` for compatibility with system tables available in Watcom SQL. These views will conflict with the Sybase SQL Server compatibility views `dbo.syscolumns` and `dbo.sysindexes` if the `-c` case sensitivity command-line switch is not used. The `-k` command-line switch causes DBINIT to omit the generation of Watcom SQL compatibility views.

**Use a transaction log mirror** A transaction log mirror is an identical copy of a transaction log, usually maintained on a separate device, for greater protection of your data. By default, SQL Anywhere does not use a mirrored transaction log. If you do wish to use a transaction log mirror, this option allows you to provide a filename. For the DBINIT command-line utility, this is the `-m` command-line switch.

**Do not use a transaction log** Creating a database without a transaction log is more economical on disk space. The transaction log is required for data replication and provides extra security for database information in case of media failure or system failure. For the DBINIT command-line utility, this is the `-n` command-line switch.

**Page size** The page size for a SQL Anywhere database can be 512, 1024, 2048 or 4096 bytes, with 1024 being the default. Other values for the size will be changed to the next larger size. Large databases usually benefit from a larger page size. For the DBINIT command-line utility, this is the `-p` command-line switch.

**Output log messages to file** Redirect log messages to the named file. For the DBINIT command-line utility, this is the `-o` command-line switch.

**Operate quietly** Do not display messages on a window. For the DBINIT command-line utility, this is the `-q` command-line switch. This option is not available from other environments.

**Set the transaction log filename** The transaction log is a file where the database engine logs all changes made by all users no matter what application system is being used. The transaction log plays a key role in backup and recovery (see "The transaction log" in the chapter "Backup and Data Recovery"), and in data replication. If the filename has no path, it is placed in the same directory as the database file. For the DBINIT command-line utility, this is the `-t` command-line switch. If you run DBINIT without specifying `-t` or `-n`, a transaction log is created with the same filename as the database file, but with extension LOG.

**Collating sequence or collation filename** The collation sequence is used for all string comparisons in the database.

The database is created with all collations listed in `SYS.SYSCOLLATION`.

If you want to use a custom collation, then you should use the Collation utility to extract the closest collation from the database, then modify the collation file and use the Initialization utility to specify the new collation. It is important to change the collation label in the custom collation file, otherwise the Initialization utility prevents the insertion of the new collation, since it will be a duplicate label.

☞ For more information on custom collating sequences, see the chapter "Database Collations".

In order to change the collation that an existing database is using, it is necessary to unload the database, create a new database using the appropriate collation, then reload the database.

For the DBINIT command-line utility, this is the `-z` command-line switch. If `-z` is specified and names one of the collations then that collation will be used by the database. If `-z` is specified but does not name one of the collations, then it is assumed that the name is a filename, and the file will be opened and the collation will be parsed from the file. The specified collation will then be used by the database. If `-z` is not specified, then the default collation is used. Normal ASCII (binary) ordering is used for the lower 128 characters. For the upper 128 characters (also called the extended characters), characters which are accented forms of a letter in the lower 128 are sorted to the same position as the unaccented form. The determination of whether or not an extended character is a letter is based upon code page 850 (multilingual code page).

The following table identifies the available collating sequence labels. All collating sequences except EBCDIC do not affect the normal ASCII character set.

<b>Label</b>	<b>Explanation</b>
437EBCDIC	(Code Page 437, EBCDIC)
437LATIN1	(Code Page 437, Latin 1)
437ESP	(Code Page 437, Spanish)
437SVE	(Code Page 437, Swedish/Finnish)
850CYR	(Code Page 850, Cyrillic)
850DAN	(Code Page 850, Danish)
850ELL	(Code Page 850, Greek)
850ESP	(Code Page 850, Spanish)
850ISL	(Code Page 850, Icelandic)
850LATIN1	(Code Page 850, Latin 1)
850LATIN2	(Code Page 850, Latin 2)
850NOR	(Code Page 850, Norwegian)
850RUS	(Code Page 850, Russian)
850SVE	(Code Page 850, Swedish/Finnish)
850TRK	(Code Page 850, Turkish)
852LATIN2	(Code Page 852, Latin 2)
852CYR	(Code Page 852, Cyrillic)
855CYR	(Code Page 855, Cyrillic)



Label	Explanation
856HEB	(Code Page 856, Hebrew)
857TRK	(Code Page 857, Turkish)
860LATIN1	(Code Page 860, Latin 1)
861ISL	(Code Page 861, Icelandic)
862HEB	(Code Page 862, Hebrew)
863LATIN1	(Code Page 863, Latin 1)
865NOR	(Code Page 865, Norwegian)
866RUS	(Code Page 866, Russian)
869ELL	(Code Page 869, Greek)
SJIS	(Japanese Shift-JIS Encoding)
EUC_JAPAN	(Japanese EUC JIS X 0208-1990 and JIS X 0212-1990 encoding)
EUC_CHINA	(Chinese GB 2312-80 Encoding)
EUC_TAIWAN	(Taiwanese Big 5 Encoding)
EUC_KOREA	(Korean KS C 5601-1992 Encoding)
UTF8	(UCS-4 Transformation Format)

**List the available collating sequences** Running the DBINIT command-line utility with the -l (L) option displays a list of available collating sequences and then stops. No database is created. The list of available collating sequences is automatically presented in Sybase Central and in the ISQL Database Tools window.

# The ISQL utility

**Syntax**                    **isql** [*switches*] [*isql-command*]

**rtsql**    [*switches*] [*isql-command*]

**Windows 3.x**            **isqlw**   [*switches*] [*isql-command*]

**syntax**                    **rtsqlw** [*switches*] [*isql-command*]

Switch	Description
-b	Do not print banner
-c " <i>keyword=value; ...</i> "	Supply database connection parameters
-k	Close window when finished (RTSQL only)
-q	Quiet mode—no windows or messages
-v	Verbose—output information on commands
-x	Syntax check only—no commands executed

**See also**                    "The database engine" on page 814

**Description**

ISQL provides the user with an interactive environment for database browsing and for sending SQL statements to SQL Anywhere.

ISQL allows you to type SQL commands, or run ISQL command files. It also provides feedback about the number of rows affected, the time required for each command, the execution plan of queries, and any error messages.

If *isql-command* is specified, then ISQL executes the command. The most common form uses the READ statement to execute an ISQL command file in batch mode. If no *isql-command* is specified, then ISQL enters the interactive mode where you can type a command into a command window.

The Windows 3.x executable name is ISQLW. You can set up Program Manager icons to start ISQLW with the same command line syntax as specified above.

RTSQL and RTSQLW are runtime SQL command processors. These programs are included in the SQL Anywhere Runtime System for Windows 3.x, 95, or NT, OS/2, or DOS. They are like ISQL except there is no interactive part; they can only run command files. By including RTSQL commands, SQL command files can be run from operating system batch or command files. RTSQL automatically loads the database engine if it is not already loaded. It unloads the database engine on exit. The command line switches are the same as those of ISQL.

**Switches**

- b** Do not print the identification banner when ISQL starts up.
- c "keyword=value; ..."** Specify connection parameters. See "Database connection parameters" in the chapter "Connecting to a Database" for a description of the connection parameters. If this option is not specified, the environment variable SQLCONNECT is used. If required connection parameters are not specified, then you are presented with a dialog to enter the connection parameters.
- k** Close the window when RTSQL is finished. This switch applies only to RTSQL in Windows 3.x, Windows 95, and Windows NT.
- q** Operate quietly. ISQL does not display any information on the screen. This is only useful if a command or command file is executed when starting ISQL.
- v** Verbose output. Information about each command is displayed as it is executed (RTSQL only).
- x** Scan commands but do not execute them. This is useful for checking long command files for syntax errors.

**Commands available in ISQL**

ISQL commands are broken into the following groups:

- ◆ Standard SQL statements that are part of the Watcom-SQL language. These commands are further broken into two groups, the data manipulation commands and the data definition commands.
- ◆ Commands that are particular to ISQL and manipulate the ISQL environment.
- ◆ Simple commands that manipulate the data window.

**ISQL Commands**

**CONFIGURE Statement** Activate the ISQL configuration window for displaying and changing ISQL options

**CONNECT Statement** Reconnect to the database engine with a different user ID and password

**DBTOOL Statement** Invoke one of the database tools

**DISCONNECT Statement** Disconnect from the database engine

**EXIT Statement** Leave ISQL

**HELP Statement** Enter the online Help facility

**INPUT Statement** Do mass input into database tables

**OUTPUT Statement** Output the current query results to a file

**PARAMETERS Statement** Specify the parameters to a command file

**QUIT Statement** Leave ISQL

**READ Statement** Execute ISQL command files

**SET CONNECTION Statement** Change the active database connection

**SET OPTION Statement** Set options that control the ISQL environment

**SYSTEM Statement** Executes an operating system command (not available in Windows 3.x).


ISQL data window  
manipulation  
commands

The data window manipulation commands are as follows (these commands are not described in the command summary section):

**CLEAR Statement** Clear the data window

**DOWN [n]** Move the current query results down n lines. The default is one line.

**UP [n]** Move the current query results up n lines. The default is one line.

 For more  
information

For detailed descriptions of SQL statements and ISQL commands, see the chapter "Watcom-SQL Language Reference".

## Starting ISQL from Sybase Central

You can start ISQL from Sybase Central in the following ways:

- ◆ Right-click a database, and select Open ISQL from the popup menu.
- ◆ Right-click a table, and select View Data from the popup menu. ISQL opens with the data in the table displayed in the Data window.
- ◆ Right-click a stored procedure, and select Test from the popup menu. ISQL opens with a test script in the Command window.

## The Log Translation utility

With the Log Translation utility you can translate a transaction log into a SQL command file.

You can access the Log Translation utility in the following ways:

- ◆ From Sybase Central, for interactive use under Windows 95 or NT.
- ◆ From the ISQL Database Tools window.
- ◆ From the ISQL DBTOOL statement.
- ◆ From the system command line, using the DBTRAN command-line utility. This is useful for incorporating into batch or command files.

### Translating a transaction log in Sybase Central

❖ **To translate a transaction log into a command file:**

- 1 Open the Database Utilities folder in the left panel.
- 2 Double-click Translate Log in the right panel. The Translate Log Wizard is displayed
- 3 Follow the instructions in the Wizard.

### Translating a transaction log from the ISQL Database Tools window

❖ **To use the Log Translation utility from the ISQL Database Tools window:**

- 1 Select Database Tools from the Window menu.
- 2 Click Translate Transaction Log to SQL on the Tools list.
- 3 Enter the transaction log filename (with path).
- 4 Click Translate, and select from the options in the dialog.
- 5 Click OK to translate the selected transaction log file to a SQL command file.

## Translating a transaction log using the DBTOOL statement

### Syntax

The syntax for the Log Translation utility from the ISQL DBTOOL statement is as follows:

```

DBTOOL TRANSLATE [ TRANSACTION ] LOG FROM logname
... [ TO sqlfile ] [ WITH ROLLBACKS ]
... | [ USERS u1, u2, ... , un ] |
    | [ EXCLUDE USERS u1, u2, ... , un ] |
... [ LAST CHECKPOINT ] [ ANSI ] [ NOCONFIRM ]
    
```

## The DBTRAN command-line utility

### Syntax

**dbtran** [*switches*] *transaction-log* [*sql-log*]

### Windows 3.x syntax

**dbtranw** [*switches*] *transaction-log* [*sql-log*]

Switch	Description
-a	Include rollback transactions in output
-f	Output from most recent transaction
-j <i>date/time</i>	Output from checkpoint prior to given date
-r	Remove uncommitted transactions (default)
-s	Produce ANSI standard SQL UPDATE transactions
-t	Include trigger-generated transactions in output
-u <i>userid,...</i>	Translate transactions for listed users
-x <i>userid,...</i>	Exclude transactions for listed users
-y	Replace file without confirmation
-z	Include trigger-generated transactions as comments

The *sql-log* filename defaults to the transaction log name with the extension SQL.

When the DBTRAN command-line utility is run, it displays the earliest log offset contained in the transaction log being translated. This can be an effective method for determining the order in which multiple log files were generated.

☞ For more information about log offset numbers please see "The SQL Remote message tracking system" in the chapter "SQL Remote Administration"

☞ For more information about the command-line switches, see "Log translation utility options", next.

## Log translation utility options

**Include uncommitted transactions** The transaction log contains any changes made before the most recent COMMIT by ANY transaction. Changes made after the most recent commit are not present in the transaction log. For the DBTRAN command-line utility, this is the -a command-line switch.

**Translate from last checkpoint only** Only those transactions completed since the last checkpoint are translated. For the DBTRAN command-line utility, this is the -f command-line switch.

**Output from checkpoint prior to given date** Only transactions from the most recent checkpoint prior to the given date and/or time are translated. The user-provided argument can be a date, time or date and time enclosed in quotes. If the time is omitted, the time is assumed to be the beginning of the day of the given date. If the date is omitted, the current day is assumed. The following is an acceptable format for the date and time: "YY/MMM/DD HH:MM". For the DBTRAN command-line utility, this is the -j date/time command-line switch.

**Do not include uncommitted transactions** Remove any transactions that were not committed. For the DBTRAN command-line utility, this is the -r command-line switch, and the default operation.

**Generate ANSI standard SQL UPDATE** For cases where there is no primary key or unique index on a table, the Translation utility generates UPDATE statements with a non-standard FIRST keyword, in case of duplicate rows. The ANSI standard UPDATE flag does not output this keyword. For the DBTRAN command-line utility, this is the -s command-line switch.

**Include transactions generated by triggers** By default, actions carried out by triggers are not included in the command file. If the matching trigger is in place in the database, then when the command file is run against the database, the trigger will carry out the actions automatically. Trigger actions should be included in the case that the matching trigger does not exist in the database against which the command file is to be run. For the DBTRAN command-line utility, this is the -t command-line switch.

**Translate transactions for listed users only** For the DBTRAN command-line utility, this is the -u command-line switch.

**Translate transactions except for listed users** For the DBTRAN command-line utility, this is the -x command-line switch.

**Operate without confirming actions** Without this option, you are prompted to confirm the replacement of an existing command file. For the DBTRAN command-line utility, this is the -y command-line switch.

**Include transactions generated by triggers as comments only** Transactions generated by triggers will be included only as comments in the output file. For the DBTRAN command-line utility, this is the -z command-line switch.



# The Open Server Gateway

The SQL Anywhere Open Server enables Sybase **dblib** and **ctlib** applications to connect to SQL Anywhere databases.

## Syntax

**dbos50** [*switches*] [*open-server-name*]

Switch	Description
<b>-c</b> <i>version</i>	Release 5.501 data type mapping compatibility
<b>-d</b> <i>database-name</i>	Name of database for connections
<b>-e</b> <i>engine-name</i>	Name of engine or client for connections
<b>-o</b> <i>log_file</i>	Output messages to file
<b>-p</b>	Run in non-preemptive mode
<b>-t</b>	Truncate the log file
<b>-v</b>	Verbose operation (extended messages)
<b>-w</b>	React to Warnings as Errors

## See also

The chapter "Using the Open Server Gateway"  
 "The Open Server Stop utility" on page 856  
 "The Open Server Information utility" on page 855.

## Description

The SQL Anywhere Open Server Gateway enables Sybase **dblib** and **ctlib** applications to connect to SQL Anywhere databases.

For SQL Anywhere databases participating in a Replication Server installation, SQL Anywhere Open Server Gateway is required at primary and replicate sites to enable Replication Server to connect to the database. At a SQL Anywhere replicate site, Replication Server must connect in order to apply changes to the database. At a SQL Anywhere primary site, Replication Server must connect for the materialization step and if asynchronous procedure calls are being carried out.

The default Open Server Gateway name is SQLAny.

## Switches

**-c** *version* The mapping of data types between Open Client applications and SQL Anywhere was modified in release 5.502. The **-c** switch is provided for Open Client applications requiring the older data type mapping behavior (available in releases 5.501 and earlier). Other forms of this switch include: **-c 5500**, **-c 5.501**, **-c 5.500**.

☞ For more information about the data type mapping of Open Server Gateway, see "Data type mappings" in the chapter "Using the Open Server Gateway".

**-d database-name** Sets the database for connections. Connections to SQL Server are made to the server, while connections to SQL Anywhere are made to an individual database. The -d switch allows Open Server Gateway to support SQL Server connection events. If no -d switch is provided, the Open Server uses the first active database on the specified engine or client for all connections.

**-e server-name** Sets the SQL Anywhere server name for connections. Connections to SQL Server are made to the server, while connections to SQL Anywhere are made to an individual database. The -d switch allows Open Server Gateway to support SQL Server connection events. If no -e switch is provided, the Open Server looks for a running engine or client and uses the first engine or client it finds for all connections.

**-o logfile** Name the log file to which messages are output. The default name for the log file is the server name with extension .WOS.

**-p** Run in non-preemptive mode. In non-preemptive mode the Open Server processes commands serially rather than concurrently.

**-t** Truncate the log file on startup.

**-v** With the -v switch set, the messages sent to the log file are also sent to the current command-line window.

**Caution**

*The -v switch is not recommended for use in a production environment, as it can have a significant impact on performance.*

**-w** With the -w switch set, the Open Server reacts to warnings as if they were errors, stopping execution. This may be useful during development, especially if the application is to run against other Open Servers also.

# The Open Server Information utility

**Syntax** `dbosinfo [switches]`

Switch	Description
<code>-s server</code>	Open Server to return information on
<code>-u userid</code>	Connection username
<code>-p password</code>	Connection password
<code>-v</code>	Verbose operation

**See also** "The Open Server Gateway" on page 853, "The Open Server Stop utility" on page 856.

**Description** The DBOSINFO command-line utility returns information about a Open Server Gateway. Currently, the only information returned is the TDS version.

**Switches**

- S server** The Open Server to return information on. The Open Server name is declared on the DBOS50 command line, and corresponds to an entry in the list of Open Servers maintained by SQLEDT .
- U userid** A SQL Anywhere user ID on a database being supported by the Open Server.
- P password** The SQL Anywhere connection password corresponding to the user ID provided in the -U command line.
- v** Extra messages are sent to the window.

## The Open Server Stop utility

**Syntax** `dbosstop [switches]`

Switch	Description
<code>-s server</code>	Open Server to stop
<code>-u userid</code>	Connection username
<code>-p password</code>	Connection password
<code>-q</code>	Operate quietly
<code>-v</code>	Verbose operation

**See also** "The Open Server Gateway" on page 853, "The Open Server Information utility" on page 855.

**Description** The DBOSSTOP command-line utility stops a Open Server Gateway.

**Switches**

**-S server** The Open Server to stop. The Open Server name is declared on the DBOS50 command line, and corresponds to an entry in the list of Open Servers maintained by SQLEDIT .

**-U userid** A SQL Anywhere user ID on a database being supported by the Open Server Gateway.

**-P password** The SQL Anywhere connection password corresponding to the user ID provided in the -U command line.

**-q** Operate quietly. No messages are sent to the log file.

**-v** Extra messages are sent to the window.

**Example** The following command stops the Open Server gateway named **primeos**, using user ID DBA, and password SQL.

```
dbosstop -S PRIMEOS -U dba -P sql
```

## The REBUILD batch or command file

**Syntax** `rebuild old-database new-database [dba-password]`

**See also** DBINIT, DBUNLOAD, ISQL

**Description** This DOS or Windows NT batch file, OS/2 command file, or QNX script uses DBUNLOAD to rebuild *old-database* into *new-database*. This is a simple script, but it helps document and automate the rebuilding process. Both database names should be specified without extensions. An extension of DB. will automatically be added.

The *dba-password* must be specified if the password to the DBA user ID in the *old-database* is not the initial password SQL.

REBUILD runs the DBUNLOAD, DBINIT, and ISQL commands with the default command line options. If you need different options, you will need to run the three steps separately.

## The SQL Remote Database Extraction utility

You can access the remote database extraction utility in the following ways:

- ◆ From Sybase Central, for interactive use under Windows 95 or NT.
- ◆ From the system command line, using the DBXTRACT command-line utility. This is useful for incorporating into batch or command files.

By default, the extraction utility runs at isolation level zero. If you are extracting a database from a running server, you should run it at isolation level 3 (see "Extraction utility options" on page 860) to ensure that data in the extracted database is consistent with data on the server. Running at isolation level 3 may hamper others' turnaround time on the server because of the large number of locks required. It is recommended that you run the extraction utility when the server is not busy.

Objects owned by  
dbo

The dbo user ID owns a set of SQL Server-compatible system objects in a SQL Anywhere database.

The Extraction utility does not unload the objects created for the dbo user ID during database creation. Changes made to these objects, such as redefining a system procedure, are lost when the data is unloaded. Any objects created by the dbo user ID since the initialization of the database are unloaded by the Extraction utility, and so these objects are preserved.

### **Extraction utility limitations**

The extraction utility is not designed to create a flawless reload script under all circumstances, but can produce a script which can be modified to the user's needs. An inconsistent database could still be created, if for example, a view referencing non-replicated tables is extracted.

## Extracting a remote database in Sybase Central

- ❖ **To extract a remote database from a running database:**
  - 1 Connect to the database.
  - 2 Right-click the database and click Extract Database in the popup menu.
  - 3 Follow the instructions in the wizard.

❖ **To extract a remote database from a database file or a running database as follows::**

- 1 Open the Database Utilities folder in the left panel.
- 2 Double-click Extract a Database in the right panel.
- 3 Follow the instructions in the wizard.

☞ For full information on extracting a remote database in Sybase Central, see the Sybase Central online Help.

## The DBXTRACT command-line utility

### Syntax

**dbxtract** [*switches*] *directory subscriber*

### Windows 3.x syntax

**dbxtracw** [*switches*] *directory subscriber*

Switch	Description
<b>-b</b>	Do not start subscriptions
<b>-c</b> " <i>keyword=value; ...</i> "	Supply database connection parameters
<b>-d</b>	Unload data only
<b>-f</b>	Extract fully qualified publications
<b>-g</b> <i>.user</i>	Specify user name as replacement for dbo
<b>-j</b> <i>nm</i>	Repeated unload of view creation statements
<b>-l</b> <i>.level</i>	Perform all extraction operations at specified isolation level
<b>-k</b>	Close window on completion
<b>-n</b>	Extract schema definition only
<b>-o</b> <i>.file</i>	Output messages to file
<b>-p</b> <i>character</i>	Escape character
<b>-q</b>	Operate quietly: do not print messages or show windows
<b>-r</b> <i>.file</i>	Specify name of generated reload ISQL command file (default "RELOAD.SQL")
<b>-u</b>	Unordered data
<b>-v</b>	Verbose messages
<b>-xf</b>	Exclude foreign keys
<b>-xp</b>	Exclude stored procedures

-xt	Exclude triggers
-xv	Exclude views
-y	Overwrite command file without confirmation

## Description

Running the extraction utility from Sybase Central carries out the following tasks related to creating and synchronizing SQL Remote subscriptions:

- ◆ Creates a command file to build a remote database containing a copy of the data in a specified publication.
- ◆ Creates the necessary SQL Remote objects, such as message types, publisher and remote user IDs, publication and subscription, for the remote database to receive messages from and send messages to the consolidated database.
- ◆ Starts the subscription at both the consolidated and remote databases.

## Inconsistent databases

### Extraction utility limitations

The extraction utility is not designed to create a flawless reload script under all circumstances, but can produce a script which can be modified to the user's needs. An inconsistent database could still be created, if for example, a view referencing non-replicated tables is extracted.

☞ For more information about the command-line switches, see "Extraction utility options", next.

## Extraction utility options

**Do not start subscriptions automatically** If this option is selected, subscriptions at the consolidated database (for the remote database) and at the remote database (for the consolidated database) must be started explicitly using the `START SUBSCRIPTION` statement for replication to begin. For the `DBXTRACT` command-line utility, this is the `-b` switch.

**Connection parameters** For a description of the connection parameters, see "Database connection parameters" in the chapter "Connecting to a Database". If the connection parameters are not specified, connection parameters from the `SQLCONNECT` environment variable are used, if set. The user ID should have DBA authority to ensure that the user has permissions on all the tables in the database.



For the DBXTRACT command-line utility, this is the `-c` command-line switch. For example, the following statement extracts a database for remote user ID `joe_remote` from the `sademo` database running on the `sample_server` server, connecting as user ID `DBA` with password `SQL`. The data is unloaded into the `C:\UNLOAD` directory.

```
dbxtract -c "eng=sample_server;dbn=sademo;  
uid=dba;pwd=sql" c:.extract joe_remote
```

**Unload the data only** If this option is selected, the schema definition is not unloaded, and publications and subscriptions are not created at the remote database. This option is for use when a remote database already exists with the proper schema, and needs only to be filled with data. For the DBXTRACT command-line utility, this is the `-d` switch.

**Extract fully qualified publications** In most cases, you do not need to extract fully qualified publication definitions for the remote database, since it typically replicates all rows back to the consolidated database anyway.

However, you may want fully qualified publications for multi-tier setups or for setups where the remote database has rows that are not in the consolidated database. For the DBXTRACT command-line utility, this is the `-f` switch.

**Repeated unload of view creation statements** If your database contains view definitions that are dependent on each other, you can use this option to avoid failure in reloading the views into a database. The option causes view creation statements to be unloaded multiple times, as specified by the count entered. This count should be small, and should correspond to the number of levels of view dependency. For the DBXTRACT command-line utility, this is the `-j` command-line switch.

**Perform extraction at a specified isolation level** The default setting is an isolation level of zero. If you are extracting a database from a running server, you should run it at isolation level 3 (see "Extraction utility options" on page 860) to ensure that data in the extracted database is consistent with data on the server. Increasing the isolation level may result in large numbers of locks being used by the extraction utility, and may restrict database use by other users. For the DBXTRACT command-line utility, this is the `-l` switch.

**Unload the schema definition only** With this definition, none of the data is unloaded. The reload file contains SQL statements to build the database structure only. You can use the `SYNCHRONIZE SUBSCRIPTION` statement to load the data over the messaging system. Publications, subscriptions, `PUBLISH` and `SUBSCRIBE` permissions are part of the schema. For the DBXTRACT command-line utility, this is the `-n` switch.

**Output messages to file** For the DBXTRACT command-line utility, this is the -o switch.

**Escape character** The default escape character (\) can be replaced by another character using this option. For the DBXTRACT command-line utility, this is the -p command-line switch.

**Operate quietly** Display no messages except errors. For the DBXTRACT command-line utility, this is the -q command-line switch. This option is not available from other environments.

**Reload filename** The default name for the reload command file is RELOAD.SQL in the current directory. For the DBXTRACT command-line utility, this is the -r switch.

**Output the data unordered** By default the data in each table is ordered by primary key. Unloads are quicker with the -u switch, but loading the data into the remote database is slower. For the DBXTRACT command-line utility, this is the -u switch.

**Verbose mode** The name of the table being unloaded and the number of rows unloaded are displayed. The SELECT statements being used to extract data from tables is also displayed, which can be useful because the SELECT statements have WHERE clauses that can lead to slow extracts, such as when indexes have not been created. For the DBXTRACT command-line utility, this is the -v switch.

**Exclude foreign key definitions** You can use this if the remote database contains a subset of the consolidated database schema, and some foreign key references are not present in the remote database. For the DBXTRACT command-line utility, this is the -xf switch.

**Exclude stored procedure** Do not extract stored procedures from the database. For the DBXTRACT command-line utility, this is the -xp switch.

**Exclude triggers** Do not extract triggers from the database. For the DBXTRACT command-line utility, this is the -xt switch.

**Exclude views** Do not extract views from the database. For the DBXTRACT command-line utility, this is the -xv switch.

**Operate without confirming actions** Without this option, you are prompted to confirm the replacement of an existing command file. For the DBXTRACT command-line utility, this is the -y switch.

# The SQL Remote Message Agent

**Syntax** `dbremote [switches] [ directory ]`

**Windows 3.x syntax** `dbremotw [switches] [ directory ]`

Switch	Description
<b>-a</b>	Do not apply received transactions
<b>-b</b>	Run in batch mode
<b>-c</b> <i>"keyword=value; ..."</i>	Supply database connection parameters
<b>-f</b> <i>minutes</i>	Receive message polling frequency (default is one minute)
<b>-k</b>	Close window on completion
<b>-l</b> <i>length</i>	Maximum message length
<b>-m</b> <i>size</i>	Maximum amount of memory to be used by DBREMOTE for building messages.
<b>-o</b> <i>file</i>	Output messages to file
<b>-p</b>	Do not purge messages
<b>-r</b>	Receive only
<b>-s</b>	Send only
<b>-t</b>	Replicate all triggers
<b>-u</b>	Process only backed up transactions
<b>-v</b>	Verbose operation
<b>-x</b>	Rename transaction log

## Description

The messaging agent sends and applies messages for SQL Remote replication, and maintains the message tracking system to ensure message delivery.

The user ID in the Message Agent command line must have either REMOTE DBA or DBA authority.

The optional *directory* parameter specifies a directory in which old transaction logs are held, so that the Message Agent has access to events from before the current log was started.

A database can have only one instance of the messaging agent connecting to it at a time. However, multiple instances of the messaging agent can connect to a single database engine or server hosting multiple databases, provided only one messaging agent connects to one database.

☞ For information on REMOTE DBA authority, see "The Message Agent and replication security" in the chapter "SQL Remote Administration". For information on log management in SQL Remote, see "Transaction log and backup management for SQL Remote" in the chapter "SQL Remote Administration".

## Switches

**-a** Process the received messages (those in the inbox) without applying them to the database. Used together with **-v** (for verbose output) and **-p** (so the messages are not purged), this flag can help detect problems with incoming messages. Used without **-p**, this flag purges the inbox without applying the messages, which may be useful if a subscription is being restarted.

**-b** Run in batch mode. In this mode, the Message Agent processes incoming messages, scans the transaction log once and processes outgoing messages, and then stops.

**-c "keyword=value; ..."** Specify connection parameters. See "Database connection parameters" in the chapter "Connecting to a Database" for a description of the connection parameters. If this option is not specified, the environment variable SQLCONNECT is used.

For example, the following statement runs DBREMOTE on a database file named C:\SQLANY50\SADEMO.DB, connecting with user ID dba and password sql:

```
dbremote -c  
"uid=dba;pwd=sql;dbf=c:\sqlany50\sademo.db"
```

The DBREMOTE command-line utility must be run by a user with REMOTE DBA authority or DBA authority.

☞ For information on REMOTE DBA authority, see "The Message Agent and replication security" in the chapter "SQL Remote Administration".

**-f minutes** Sets the frequency of incoming message checking when DBREMOTE is used in continuous mode (not batch mode). Specifies in minutes how often the source of messages is polled. Values greater than the SEND EVERY option specified in the GRANT REMOTE SQL statement cause DBREMOTE to check for incoming messages once after each send cycle. If this value is not set, a default polling frequency of one minute is used.

**-k** Close window on completion. This flag is valid for Windows and Windows NT only.

**-l length** Specifies the maximum length of message to be sent, in bytes. The default is 51200 (50K). Longer transactions are split into more than one message.

**-m size** Specifies a maximum amount of memory to be used by DBREMOTE for building messages. When the memory usage is exceeded, messages are flushed (before being full). The allowed size can be specified as *n* (in bytes), *nK*, or *nM*. The default is 2048K (2M).

This is provided for customers considering a single consolidated database for thousands of remote databases.

**-o file** Append output to a log file. Default is to send output to the screen.

**-p** Process the messages without purging them.

**-r** Receive messages only.

**-s** Send messages only.

**-t** All trigger actions are replicated. If you do use this switch, you must ensure that the trigger actions are not carried out twice at remote databases, once by the trigger being fired at the remote site, and once by the explicit application of the replicated actions from the consolidated database.

To ensure that trigger actions are not carried out twice, you can wrap an IF CURRENT REMOTE USER IS NULL ... END IF statement around the body of the triggers.

**-u** Process only transactions that have been backed up. This switch prevents DBREMOTE from processing transactions contained in the live transaction log. Transactions in the live, or active transaction log are not processed until that log file is backed up. Outgoing transactions and confirmation of incoming messages are only processed for those transactions that are not in the live transaction log.

**-v** Verbose output. This switch displays the SQL statements contained in the messages to the screen and, if the -o switch is used, to a log file.

**-x** Renames transaction log after it has been scanned for outgoing messages. Since replication can have the effect of backing up a database, in many cases this switch eliminates the need to run DBBACKUP on the remote computer (or renaming the transaction log when the engine is shut down).

## **Registry settings**

SQL Remote uses several registry settings (in Windows 95 and NT) or initialization file settings (Windows 3.x and OS/2) to control aspects of message link behavior.

In the registry, the settings are stored in the Current User, under Software►Sybase►SQL Anywhere►Sybase SQL Anywhere 5.0►SQL Remote. In Windows 3.x and OS/2, they are stored in the SQLANY.INI initialization file.

☞ For a listing of registry settings, see "Message link control parameters" in the chapter "SQL Remote Administration".

## The Stop utility

The Stop utility stops a SQL Anywhere standalone engine, network server, or SQL Anywhere Client.

All memory used by the database engine is returned to the system for use by other applications.

The Stop utility is a command-line utility only. In Windowing environments, you can stop a database engine or server by clicking Close on the engine window or choosing Exit from the File menu on the server window.

## The DBSTOP command-line utility

### Syntax

**dbstop** [*switches*] { *name* | CLIENT }

### Windows 3.x syntax

**dbstopw** [*switches*] { *name* | CLIENT }

Parameter	Description
<b>-q</b>	Quiet mode—do not print messages
<b>-s</b> " <i>keyword=value; ...</i> "	Supply database connection parameters
<b>-x</b>	Do not stop if there are active connections
<i>name</i>	Engine or server name

In DOS, it is not necessary to specify a *name*, because only one database engine or client can be running.

In QNX, DBSTOP can shut down a server on any node on the network. The *name* is necessary to specify the name of the server you wish to stop.

The DBSTOP command is not necessary after the database engine has been started by ISQL. ISQL automatically unloads the database engine when it disconnects.

DBSTOP CLIENT stops the SQL Anywhere Client only.

☞ For more information about the command-line switches, see "Stop utility options", next.

## Stop utility options

**-q** Operate quietly. Do not print a message if the database was not running.

**-s** When stopping a database server, you must supply a connection string with a user ID that has permissions to stop the server or engine. By default, DBA permission is required on the database server, and all users can shut down a standalone engine, but the **-gk** switch on the engine or server can be used to change this.

☞ For a description of the connection parameters, see "Database connection parameters" in the chapter "Connecting to a Database". To shut down a database server, the connection string must include an **AGENT** parameter in addition to the usual connection parameters. **AGENT** must be set to the value **SERVER** to shut down a server. A sample command line is:

```
dbstop -s "uid=dba;pwd=sql;  
agent=server;unconditional=true"
```

**-x** Do not stop the engine if there are still active connections to the engine.

### QNX switch

**-p password** Specify a password to unlock the server. If the server's keyboard is locked and no password is specified or the specified password is incorrect, then the server will not shut down.



## The Transaction Log utility

With the Transaction Log utility you can display or change the name of the transaction log or transaction log mirror associated with a database. You can also stop a database from maintaining a transaction log or mirror, or start maintaining a transaction log or mirror.

A transaction log mirror is a duplicate copy of a transaction log, maintained by a database in tandem.

The name of the transaction log is first set when the database is initialized. The Transaction Log utility works with database files or with write files. The database engine must not be running on that database when the transaction log filename is changed (an error message is displayed if it is).

You can access the Transaction Log utility in the following ways:

- ◆ From Sybase Central, for interactive use under Windows 95 or NT
- ◆ From the ISQL Database Tools window.
- ◆ From the ISQL DBTOOL statement.
- ◆ From the system command line, using the DBLOG command-line utility

### Changing a log file name from Sybase Central

❖ **To change a transaction log file name:**

- 1 Open the Database Utilities folder in the left panel.
- 2 Double-click Change Log File in the right panel. The Transaction Log Wizard is displayed
- 3 Follow the instructions in the Wizard.

🔗 For full information on changing a log file name from Sybase Central, see the Sybase Central online Help.

## Changing a log file name from the ISQL Database Tools window

❖ **To use the Translation Log utility from the ISQL Database Tools window to change a log filename:**

- 1 Select Database Tools from the Window menu.
- 2 Click Change Transaction Log Name on the Tools list.
- 3 Enter the database filename (with path).
- 4 Click Change, and select from the options on the dialog.
- 5 Click OK to change the name of the transaction log, or to run without a transaction log.

## Changing a log filename from the DBTOOL statement

**Syntax**

```
DBTOOL ALTER DATABASE name  
... NO [ TRANSACTION ] LOG  
| SET [ TRANSACTION ] LOG TO filename
```

You can rename the database filename as well as the transaction log name from the DBTOOL statement.

## The DBLOG command-line utility

**Syntax**

```
dblog [switches] database-file
```

**Windows 3.x  
syntax**

```
dblogw [switches] database-file
```

Switch	Description
-g <i>n</i>	Sets the generation number to <i>n</i>
-il	Ignores the LTM truncation offset stored in the database
-ir	Ignores the SQL Remote truncation offset stored in the database
-m <i>mirror-name</i>	Set transaction log mirror name
-n	No longer use a transaction log
-o	Output messages to file
-q	Quiet mode—do not print messages

<b>-r</b>	No longer use a transaction log mirror
<b>-t</b> <i>log-name</i>	Set transaction log name

☞ For more information about the command-line switches, see "Transaction log utility options", next.

## Transaction log utility options

**Set the generation number** This option is for use when using the SQL Anywhere Log Transfer Manager in order to participate in a Replication Server installation. It can be used after a backup is restored to set the generation number. It performs the same function as the following Replication Server function:

```
dbcc settrunc( 'ltm', 'gen_id', n ).
```

For information on generation numbers and dbcc, see your Replication Server documentation. For the DBLOG command-line utility, this is the **-g** command-line switch.

**Ignore the LTM truncation offset** This option is for use when using the SQL Anywhere Log Transfer Manager in order to participate in a Replication Server installation. It performs the same function as the following Replication Server function:

```
dbcc settrunc( 'ltm', 'gen_id', n ).
```

For information on dbcc, see your Replication Server documentation. For the DBLOG command-line utility, this is the **-il** command-line switch.

**Ignore the SQL Remote truncation offset** This option is for use when using the SQL Anywhere Log Transfer Manager in order to participate in a Replication Server installation and a SQL Remote installation. It resets the offset that is kept for the purposes of the DELETE\_OLD\_LOGS option, allowing transaction logs to be deleted when they are no longer needed. For the DBLOG command-line utility, this is the **-ir** command-line switch.

**Set the name of the transaction log mirror file** This option sets a filename for a new transaction log mirror. If the database is not currently using a transaction log mirror, it starts using one. If the database is already using a transaction log mirror, it changes to using the new filename as its transaction log mirror. For the DBLOG command-line utility, this is the **-m** command-line switch.

**No longer use a transaction log** Stop using a transaction log. Without a transaction log, the database will no longer be able to participate in data replication or use the transaction log in data recovery. For the DBLOG command-line utility, this is the `-n` command-line switch.

**Output log messages to file** Redirect log messages to the named file. For the DBLOG command-line utility, this is the `-o` command-line switch.

**Operate quietly** Do not display messages on a window. For the DBLOG command-line utility, this is the `-q` command-line switch. This option is not available from other environments.

**No longer use a transaction log mirror** For databases maintaining a mirrored transaction log, this option changes their behavior to maintain only a single transaction log. For the DBLOG command-line utility, this is the `-r` command-line switch.

**Set the name of the transaction log file** This option sets a filename for a new transaction log. If the database is not currently using a transaction log, it starts using one. If the database is already using a transaction log, it changes to using the new filename as its transaction log. For the DBLOG command-line utility, this is the `-t` command-line switch.

## The Uncompression utility

With the Uncompression utility you can expand a compressed database file created by the Compression utility. The Uncompression utility reads the given compressed database file and creates an uncompressed database file.

The Uncompression utility does not uncompress files other than the main database file (dbspace files).


You can access the Uncompression utility in the following ways:

- ◆ From Sybase Central, for interactive use under Windows 95 or NT.
- ◆ From the ISQL Database Tools window.
- ◆ From the ISQL DBTOOL statement.
- ◆ From the system command line, using the DBEXPAND command-line utility. This is useful for incorporating into batch or command files.

### Uncompressing a database in Sybase Central

❖ **To uncompress a compressed database file:**

- 1 Open the Database Utilities folder in the left panel.
- 2 Double-click Uncompress Database in the right panel. The Uncompress Wizard is displayed.
- 3 Follow the instructions in the Wizard.

 For full information on uncompressing a database in Sybase Central, see the Sybase Central online Help.

### Uncompressing a database from the ISQL Database Tools window

❖ **To use the Uncompression utility from the ISQL Database Tools window:**

- 1 Select Database Tools from the Window menu.
- 2 Click Uncompress Database on the Tools list.
- 3 Enter the database filename (with path).

- 4 Click Uncompress, and enter a filename to use for the uncompressed database.
- 5 Click OK to uncompress the selected database file.

## Uncompressing a database using the DBTOOL statement

**Syntax** The syntax for the Uncompression utility from the ISQL DBTOOL statement is as follows:

```
DBTOOL UNCOMPRESS DATABASE compressed-database-file  
... [ TO filename ] [ NOCONFIRM ]
```

**Example** The following statement uncompresses the compressed database file C:\TEMP.CDB, creating the database file C:\TEMP.DB

```
DBTOOL UNCOMPRESS DATABASE 'c:\temp.cdb'
```


## The DBEXPAND command-line utility

**Syntax** **dbexpand** [*switches*] *compressed-database-file* [*database-file*]

**Windows 3.x syntax** **dbexpandw** [*switches*] *compressed-database-file* [*database-file*]

Switch	Description
-q	Operate quietly—do not print messages
-y	Erase existing output file without confirmation

The input filename extension defaults to CDB. The output filename (with extension) must not have the same name as the input filename (with extension).

 For more information about the command-line switches, see "Uncompression utility options", next.

## Uncompression utility options

**Output log messages to file** Redirect log messages to the named file. For the DBEXPAND command-line utility, this is the -o option.

**Operate quietly** Do not display messages on a window. For the DBEXPAND command-line utility, this is the `-q` command-line switch. This option is not available from other environments.

**Operate without confirming actions** Without this option, you are prompted to confirm the replacement of an existing database file. For the DBEXPAND command-line utility, this is the `-y` command-line switch.

## The Unload utility

With the Unload utility you can unload a database and put data files into a set of files in a named directory. The Unload utility creates the ISQL command file RELOAD.SQL to rebuild your database from scratch. It also unloads all of the data in each of your tables in comma delimited format into files in the specified directory. Binary data is properly represented with escape sequences.

You can access the Unload utility in the following ways:

- ◆ From Sybase Central, for interactive use under Windows 95 or NT.
- ◆ From the ISQL Database Tools window.
- ◆ From the ISQL DBTOOL statement.
- ◆ From the system command line, using the DBUNLOAD command-line utility. This is useful for incorporating into batch or command files.

The Unload utility should be run from a DBA user ID. This is the only way you can be sure of having privileges to unload all the data. Also, the RELOAD.SQL file should be run from the DBA user ID. (Usually it will be run on a new database where the only user ID is DBA with password SQL.)

Objects owned by  
dbo

The dbo user ID owns a set of SQL Server-compatible system objects in a SQL Anywhere database.

The Unload utility does not unload the objects created for the dbo user ID during database creation. Changes made to these objects, such as redefining a system procedure, are lost when the data is unloaded. Any objects created by the dbo user ID since the initialization of the database are unloaded by the Unload utility, and so these objects are preserved.

## Unloading a database from Sybase Central

### ❖ To unload a running database:

- 1 Connect to the database.
- 2 Right-click the database and click Unload in the popup menu. The Unload Database Wizard is displayed.
- 3 Follow the instructions in the Wizard.



❖ **To unload a database file or a running database:**

- 1 Open the Database Utilities folder in the left panel.
- 2 Double-click Unload Database in the right panel. The Unload Database Wizard is displayed.
- 3 Follow the instructions in the Wizard.

🔗 For full information on unloading a database from Sybase Central, see the Sybase Central online Help.

## Unloading a database from the ISQL Database Tools window

❖ **To use the Unload utility from the ISQL Database Tools window:**

- 1 Select Database Tools from the Window menu.
- 2 Click Unload Database on the Tools list.
- 3 Enter a user ID and password to use when connecting to the database.
- 4 For a running database, enter a database name and server name (if more than one is running). For a database file, enter the filename (with path) and optionally a start line to specify command-line switches for the database engine or SQL Anywhere Client.
- 5 Click Unload, and enter a filename to use for the command file and a directory in which to hold the data. Select from the other options in the dialog.
- 6 Click OK to unload the database.

## Unloading a database from the DBTOOL statement

### Syntax

The syntax for the Unload utility from the ISQL DBTOOL statement is as follows:

```

DBTOOL UNLOAD TABLES TO directory
... [ RELOAD FILE TO filename ]
... [ DATA ]
    | [ SCHEMA ]
... [ UNORDERED ] [ VERBOSE ] USING connection-string

```

## The DBUNLOAD command-line utility

**Syntax** `dbunload [switches] directory [ table-name-list ]`

**Windows 3.x syntax** `dbunloaw [switches] directory [ table-name-list ]`

Switch	Description
<code>-c "keyword=value; ..."</code>	Supply database connection parameters
<code>-d</code>	Unload data only
<code>-e</code>	No data output for listed tables
<code>-g userid</code>	Specify user name as replacement for dbo
<code>-i</code>	Data output for listed tables only
<code>-j nnn</code>	Repeated unload of view creation statements
<code>-n</code>	No data—schema definition only
<code>-p</code>	Escape character for external unloads
<code>-q</code>	Quiet mode—no windows or messages
<code>-r reload-file</code>	Specify name and directory of generated reload ISQL command file (default RELOAD.SQL)
<code>-u</code>	Unordered data
<code>-v</code>	Verbose messages
<code>-x</code>	External unload (when client and server are on different machines)
<code>-y</code>	Replace command file without confirmation

In the default mode the directory used by DBUNLOAD to hold the data is relative to the database engine or server, not to the current directory of the user. For details of how to supply a filename and path in this mode, see "UNLOAD TABLE statement" in the chapter "Watcom-SQL Statements". If the `-x` switch is used, the directory is relative to the current directory of the user. The RELOAD.SQL command file is always relative to the current directory of the user, regardless of whether `-x` is used.

If no list of tables is supplied, the whole database is unloaded. If a list of tables is supplied, only those tables are unloaded.

☞ For more information about the command-line switches, see "Unload utility options", next.

## Unload utility options

**Connection parameters** For a description of the connection parameters, see "Database connection parameters" in the chapter "Connecting to a Database". If the connection parameters are not specified, connection parameters from the SQLCONNECT environment variable are used, if set. The user ID should have DBA authority to ensure that the user has permissions on all the tables in the database.

For the DBUNLOAD command-line utility, this is the `-c` command-line switch. For example, the following statement unloads the `sademo` database running on the `sample_server` server, connecting as user ID `DBA` with password `SQL`. The data is unloaded into the `C:\UNLOAD` directory.

```
dbunload -c
"eng=sample_server;dbn=sademo;uid=dba;pwd=sql"
c:\unload
```

**Unload data only** With this option, none of the database definition commands are generated (`CREATE TABLE`, `CREATE INDEX`, and so on); `RELOAD.SQL` contains statements to reload the data only. For the DBUNLOAD command-line utility, this is the `-d` command-line switch.

**No data output for listed tables** For the DBUNLOAD command-line utility, this is the `-e` command-line switch. This is not accessible from other environments. By default, the optional table-list defines the tables to be unloaded. If you wish to unload almost all the tables in the database, the `-e` command-line switch unloads all tables except the specified tables.

**User ID for dbo user** SQL Anywhere contains a set of system views that mimic the system tables of Sybase SQL Server. By default, the owner of these views is the user ID `dbo`, which is the same as the owner of the SQL Server system tables, but you may have configured this differently when creating or upgrading your database. If the SQL Server-compatibility system views in your database are owned by a user other than `dbo`, specify that user ID in this option. Views and procedures owned by the user ID you specify are not unloaded. For the DBUNLOAD command-line utility, this is the `-g` command-line switch.

**Data output for listed tables only** For the DBUNLOAD command-line utility, this is the `-i` command-line switch. This is not accessible from other environments. This switch is the default option, and causes only the supplied table-list to be unloaded.

**Repeated unload of view creation statements** If your database contains view definitions that are dependent on each other, you can use this option to avoid failure in reloading the views into a database. The option causes view creation statements to be unloaded multiple times, as specified by the count entered. This count should be small, and should correspond to the number of levels of view dependency. For the DBUNLOAD command-line utility, this is the `-j` command-line switch.

**Unload schema definition only** With this option, none of the data in the database is unloaded; RELOAD.SQL contains SQL statements to build the structure of the database only. For the DBUNLOAD command-line utility, this is the `-n` command-line switch.

**Escape character** The default escape character (`\`) for external unloads (DBUNLOAD `-x` switch) can be replaced by another character using this option. For the DBUNLOAD command-line utility, this is the `-p` command-line switch. This option is not available from other environments.

**Operate quietly** Display no messages except errors. For the DBUNLOAD command-line utility, this is the `-q` command-line switch. This option is not available from other environments.

**Specify reload filename** Modify the name and directory of the generated reload ISQL command file. The default is RELOAD.SQL in the current directory. For the DBUNLOAD command-line utility, this is the `-r` command-line switch.

**Output the data unordered** Normally the data in each table is ordered by the primary key. Use this option if you are unloading a database with a corrupt index so that the corrupt index is not used to order the data. For the DBUNLOAD command-line utility, this is the `-u` command-line switch.

**Enable verbose mode** The table name of the table currently being unloaded and how many rows have been unloaded is displayed. For the DBUNLOAD command-line utility, this is the `-v` command-line switch. This option is not available from other environments.

**Use external unloading** The Unload utility uses the UNLOAD TABLE statement by default. If the external unloading option is specified, the ISQL OUTPUT statement is used to extract the data, and the generated reload command file uses the INPUT INTO statement to reload the data. For the DBUNLOAD command-line utility, this is the `-x` command-line switch.

The default mode is faster than the external unloading mode. However, in the default mode the directory used by the Unload utility to hold the data is relative to the database server, not to the current directory of the user. The RELOAD.SQL command file is always relative to the current directory of the user, regardless of whether external unloading is used.

☞ For more information on filenames and paths for the Unload utility, see "UNLOAD TABLE statement" in the chapter "Watcom-SQL Statements".

**Operate without confirming actions** Without this option, you are prompted to confirm the replacement of an existing command file. For the DBUNLOAD command-line utility, this is the -y command-line switch.

## Rebuilding a database

To unload a database, start the database engine with your database, and run the Unload utility with the DBA user ID and password.

To reload a database, you need to create a new database and then run the generated RELOAD.SQL command file through ISQL or RTSQL.

In Windows 95 and Windows NT, there is a file (REBUILD.BAT) included with SQL Anywhere that automates the unload and reload process. In Windows 3.x, rebuilding can be done from the ISQL Database Tools window.

In DOS, OS/2, Windows 95 and NT, and QNX, there is a file (REBUILD.BAT, REBUILD.CMD, or rebuild) included with SQL Anywhere that automates the unload and reload process. In Windows 3.x, rebuilding can be done from the ISQL Database Tools window.

## The Upgrade utility

The Upgrade utility upgrades a database created with Watcom SQL 3.2 or Watcom SQL 4.0 to the SQL Anywhere 5.0 format. While SQL Anywhere 5.0 does run against a database created with Watcom SQL 3.2 or Watcom SQL 4.0, some of the features introduced since the version that created the database are unavailable unless the database is upgraded.

For people switching from release 3.2 or 4.0 to the current release, the Upgrade utility upgrades their databases without having to unload and reload them.

If you wish to use replication on an upgraded database, you must also archive your transaction log and start a new one on the upgraded database.


You can access the Upgrade utility in the following ways:

- ◆ From Sybase Central, for interactive use under Windows 95 or NT.
- ◆ From the system command-line, using the DBUPGRAD command-line utility.

## Upgrading a database from Sybase Central

### ❖ To upgrade a database:

- 1 Open the Database Utilities folder in the left panel.
- 2 Double-click Upgrade Database in the right panel. The Upgrade Database Wizard is displayed.
- 3 Follow the instructions in the Wizard.

 For full information on upgrading a database from Sybase Central, see the Sybase Central online Help.

## Upgrading databases too old for the Upgrade utility

### ❖ To upgrade a database created with a version of SQL Anywhere that is too old to be upgraded:

- 1 Unload the database using the Unload utility.
- 2 Create a database with the name you wish to use for the upgraded version, using the Initialization utility.

- 3 Connect to the new database from ISQL as the DBA user ID, and read the reload.sql command file to build the new database.

## The DBUPGRAD command-line utility

**Syntax** `dbupgrad [switches]`

**Windows 3.x syntax** `dbupgrdw [switches]`

Switch	Description
<code>-c "keyword=value; ..."</code>	Supply database connection parameters
<code>-g userid</code>	User ID to use as replacement for dbo
<code>-k</code>	Close window on completion
<code>-q</code>	Quiet mode—no windows or messages

☞ For more information about the command-line switches, see "Upgrade utility options", next.

## Upgrade utility options

**Connection parameters** For a description of the connection parameters, see "Database connection parameters" in the chapter "Connecting to a Database". If the connection parameters are not specified, connection parameters from the SQLCONNECT environment variable are used, if set. The user ID must have DBA authority.

For the DBUPGRAD command-line utility, this is the `-c` command line switch. For example, the following command upgrades a database called sample40 to a 5.0 format, connecting as user dba with password sql:

```
dbupgrad -c
"uid=dba;pwd=sql;dbf=c:\wsq140\sample40.db"
```

The DBUPGRAD command-line utility must be run by a user with DBA authority.

**User ID for dbo user** SQL Anywhere contains a set of system views that mimic the system tables of Sybase SQL Server. By default, the owner of these views is the user ID **dbo**, which is the same as the owner of the SQL Server system tables. If you already have a user ID named **dbo**, or you wish to use that user ID for other purposes, this option allows you to provide an alternative user ID for the owner of the SQL Server-like system views. For the DBUPGRAD command-line utility, this is the **-g** command-line switch.

**Close window on completion** Once the upgrade is completed, the message window is closed. For the DBUPGRAD command-line utility, this is the **-k** command line switch. This option is not available from other environments.

**Operate quietly** Do not display messages on a window For the DBUPGRAD command-line utility, this is the **-q** command line switch. This option is not available from other environments.



## The Validation utility

With the Validation utility you can validate all indexes and keys on some or all of the tables in the database. The Validation utility scans the entire table, and looks up each record in every index and key defined on the table.

This utility can be used in combination with regular backups (see the chapter "Backup and Data Recovery") to give you confidence in the security of the data in your database.

You can access the Validation utility in the following ways:

- ◆ From Sybase Central, for interactive use under Windows 95 or NT.
- ◆ From the ISQL Database Tools window.
- ◆ From the ISQL DBTOOL statement.
- ◆ From the system command line, using the DBVALID command-line utility. This is useful for incorporating into batch or command files.

☞ For more information

For more information on validating tables, see "VALIDATE TABLE statement" in the chapter "Watcom-SQL Statements".

## Validating a database from Sybase Central

### ❖ To validate a running database:

- 1 Connect to the database.
- 2 Right-click the database and click Validate in the popup menu.

### ❖ To validate an individual table:

- 1 Connect to the database.
- 2 Locate the table you wish to validate.
- 3 Right-click the table and click Validate in the popup menu.

☞ For full information on validating a database from Sybase Central, see the Sybase Central online Help.

## Validating a database from the ISQL Database Tools window

### ❖ To use the Validation utility from the ISQL Database Tools window:

- 1 Select Database Tools from the Window menu.
- 2 Click Check Database Integrity on the Tools list.
- 3 Enter a user ID and password to use when connecting to the database.
- 4 For a running database, enter a database name and server name (if more than one is running). For a database file, enter the filename (with path) and optionally a start line to specify command line switches for the database engine or SQL Anywhere Client.
- 5 Click Check.
- 6 Enter a list of tables to validate, if you do not wish to validate the entire database, and click OK.

## Using the Validation utility from the DBTOOL statement

The syntax for the Validation utility from the ISQL DBTOOL statement is as follows:

```
DBTOOL VALIDATE TABLES [ t1, t2, ..., tn ]  
... USING connection-string
```

## Using the DBVALID command line utility

**Syntax** `dbvalid [switches] [table-name,...]`

**Windows 3.x syntax** `dbvalidw [switches] [table-name,...]`

Switch	Description
-c "keyword=value; ..."	Supply database connection parameters
-q	Quiet mode—do not print messages

☞ For more information about the command-line switches, see "Validation utility options", next.

## Validation utility options

**Connection parameters** For a description of the connection parameters, see "Database connection parameters" in the chapter "Connecting to a Database". If the connection parameters are not specified, connection parameters from the SQLCONNECT environment variable are used, if set. The user ID must have DBA authority.

For the DBVALID command-line utility, this is the `-c` command line switch. For example, the following validates the sample database, connecting as user `dba` with password `sql`:

```
dbvalid -c  
"uid=dba;pwd=sql;dbf=c:\sqlany50\sademo.db"
```

**Operate quietly** Do not display messages on a window. For the DBVALID command-line utility, this is the `-q` command line switch. This option is not available from other environments.

## The Write File utility

The Write File utility is used to manage database write files. A **write file** is a file attached to a particular database. All changes are written into the write file, leaving the database file unchanged.

### Using write files for development

Write files can be used effectively for testing when you do not wish to modify the production database. They can also be used in network and student environments where read-only access to a database is desired, or when you distribute on CD-ROM a database you wish users to be able to modify.

### Compressed databases

If you are using a compressed database, then you must use a write file; compressed database files cannot be modified directly. The write file name is then used in place of the database name when connecting to the database or when loading a database on the database engine or server command line.

#### ❖ To access the Write File utility:


- ◆ From Sybase Central, for interactive use under Windows 95 or NT.
- ◆ From the ISQL Database Tools window.
- ◆ From the ISQL DBTOOL statement.
- ◆ From the system command line, using the DBWRITE command-line utility. This is useful for incorporating into batch or command files.

The Write File utility runs against a database file. The database must not be running on an engine when you run the Write File utility.

## Creating a write file from Sybase Central

#### ❖ To create a write file for a database:

- 1 Open the Database Utilities folder in the left panel.
- 2 Double-click Create Write File in the right panel. The Write File Wizard is displayed.
- 3 Follow the instructions in the Wizard.

 For full information on creating a write file from Sybase Central, see the Sybase Central online Help.

## Creating a write file from the ISQL Database Tools window

### ❖ To use the Write File utility from the ISQL Database Tools window:

- 1 Select Database Tools from the Window menu.
- 2 Click Create Write File on the Tools list.
- 3 Enter the database filename (with path).
- 4 Click Create, and enter a filename to use for the write file and for the write file transaction log.
- 5 Click OK to create the write file.

## Creating a write file using the DBTOOL statement

### Syntax

The syntax for the Write File utility from the ISQL DBTOOL statement is as follows:

```
DBTOOL CREATE WRITEFILE filename FOR DATABASE filename
... [ [ TRANSACTION ] LOG TO logname ] [ NOCONFIRM ]
```

## The DBWRITE command-line utility

### Syntax

```
dbwrite [switches] database-file [write-name]
```

### Windows 3.x syntax

```
dbwritew [switches] database-file [write-name]
```

Switch	Description
-c	Create a new write file
-d <i>database-file</i>	Point a write file to a different database
-f <i>database-file</i>	Force write file to point at file
-m <i>mirror-name</i>	Set transaction log mirror name
-o <i>file</i>	Output messages to file
-q	Quiet mode—do not print messages
-s	Report write file status (default)
-t <i>log-name</i>	Set transaction log name
-y	Erase/replace old files without confirmation

If any changes are made to the original database (not using the write file), the write file will no longer be valid. This happens if you start the engine using the original database file and make a modification to it. It can be made valid again by the command:

```
dbwrite -c db-name write-name
```

However, this deletes all changes recorded in the write file.

The *log-name* and *mirror-name* parameters are only used when creating a new write file. The *write-name* parameter is only used with the *-c* and *-d* parameters. Note that the *db-name* parameter must be specified before the *write-name* parameter.

☞ For more information about the command-line switches, see "Write file utility options", next.

## Write file utility options

**Create a new write file** If an existing write already exists, any information in the old write file will be lost. If no write filename is specified on the command line, the write filename will default to the database name with the extension WRT. If no transaction log name is specified, the log filename will default to the database name with the extension WLG. For the DBWRITE command-line utility, this is the *-c* command-line switch.

**Change the database file to which an existing write file points** If a database file is moved to another directory, or renamed, this option allow you to maintain the link between the write file and the database file. For the DBWRITE command-line utility, this is the *-d* command-line switch. The option is not available in other environments.

**Force a write file to point to a file** For the DBWRITE command-line utility, this is the *-f* command-line switch. The option is not available in other environments. This option is for use when a write file is being created and the database file is held on a Novell NetWare or other network path, for operating systems on which they cannot be entered directly. By providing the full Novell path name for the database file, (for example: SYS\SADemo.DB), dependencies on local mappings of the NetWare path can be avoided. Unlike the option to change the database file pointed to, no checking is done on the specified path.

**Operate quietly** Do not display messages on a window. For the DBWRITE command-line utility, this is the *-q* command-line switch. This option is not available from other environments.

**Report write file status only** This displays the name of the database that the write file points to. For the DBWRITE command-line utility, this is the -s command-line switch. The option is not available in other environments.

**Operate without confirming actions** Without this option, you are prompted to confirm the replacement of an existing database file. For the DBWRITE command-line utility, this is the -y command-line switch.

# The SQL Preprocessor

The SQL preprocessor processes a C or C++ program with embedded SQL before the compiler is run.

## Syntax

**sqlpp** [*switches*] *sql-filename* [*output-filename*]

Switch	Description
<b>-c</b>	Favor code size
<b>-d</b>	Favor data size
<b>-e level</b>	Flag non-conforming SQL syntax as an error
<b>-f</b>	Put far keyword on generated static data
<b>-h line-width</b>	Limit maximum line length of output
<b>-l userid, pswd</b>	Logon identification
<b>-n</b>	Line numbers
<b>-o operating-sys</b>	Target operating system specification DOS, DOS32, DOS286, WINDOWS, WIN32 OS232, WINNT, NETWARE, QNX32(default is DOS, OS232, or WINNT)
<b>-q</b>	Quiet mode—do not print banner
<b>-r</b>	generate reentrant code
<b>-s string-len</b>	Maximum string constant length for compiler
<b>-w level</b>	Flag non-conforming SQL syntax as a warning
<b>-z sequence</b>	Specify collation sequence

## See also

"The C language SQL preprocessor" in the chapter "The Embedded SQL Interface"

## Description

The SQL preprocessor processes a C or C++ program with embedded SQL before the compiler is run. SQLPP translates the SQL statements in the *input-file* into C language source that is put into the *output-file*. The normal extension for source programs with embedded SQL is SQC. The default output filename is the **sql-filename** with an extension of C. If the **sql-filename** has a C. extension, then the output filename extension is CC by default.

## Switches

**-c** Generate code that will favor code size and execution speed over data space size. Statically initialized data structures will be used as much as possible. This is the default.



**-d** Generate code that will reduce data space size. Data structures will be reused and initialized at execution time before use. This increases code size.

**-e level** This option flags any Embedded SQL that is not part of a specified set of SQL/92 as an error.

The allowed values of *level* and their meanings are as follows:

- ◆ **e** flag syntax that is not entry-level SQL/92 syntax
- ◆ **i** flag syntax that is not intermediate-level SQL/92 syntax
- ◆ **f** flag syntax that is not full-SQL/92 syntax
- ◆ **w** allow all supported syntax

**-f** Put the *far* keyword in front of preprocessor generated data. This may be required in conjunction with the Borland C++ compiler for the large memory model. By default, all static data is put in the same segment. Adding the *far* keyword will force static data into different segments. (By default, WATCOM C and Microsoft C place data objects bigger than a threshold size in their own segment.)

**-h num** Limits the maximum length of lines output by SQLPP to *num*. The continuation character is a backslash (\), and the minimum value of *num* is ten.

**-l userid, password** The named *userid* and *password* will be used for authorization of static SQL statements (see "Authorization" in the chapter "The Embedded SQL Interface").

**-n** Generate line number information in the C file. This consists of *#line* directives in the appropriate places in the generated C code. If the compiler you are using supports the *#line* directive, this switch will make the compiler report errors on line numbers in the SQC file (the one with the Embedded SQL) as opposed to reporting errors on line numbers in the C file generated by the SQL preprocessor. Also, the *#line* directives will indirectly be used by the source level debugger so that you can debug while viewing the SQC source file.

**-o operating-sys** Specify the target operating system. Note that this option must match the operating system you will be running the program in. A reference to a special symbol will be generated in your program. This symbol is defined in the interface library. If you use the wrong operating system specification or the wrong library, an error will be detected by the linker. The supported operating systems are:

**DOS** MS-DOS or DR-DOS

**DOS32** 32-bit DOS extended program

**DOS286** 286 DOS extended program

**WINDOWS** Microsoft Windows 3.x

**WIN32** 32-bit Microsoft Windows

**OS232** 32-bit OS/2

**WINNT** Microsoft Windows NT

**NETWARE** Novell NetWare

**QNX** 16-bit QNX

**QNX32** 32-bit QNX

The default is DOS for the DOS version of SQLPP, OS232 for the OS/2 version, WINNT for the Windows NT version and QNX for the QNX version.

**-r** Generate reentrant code. This statement is only necessary when you are writing code that is reentrant (see "Multi-Threaded or Reentrant Code" in the chapter "The Embedded SQL Interface").

**-q** Operate quietly. Do not print the banner.

**-s string-len** Set the maximum size string that the preprocessor will put into the C file. Strings longer than this value will be initialized using a list of characters ('a','b','c', etc). Most C compilers have a limit on the size of string literal they can handle. This option is used to set that upper limit. The default value is 500.

**-w level** This option flags any Embedded SQL that is not part of a specified set of SQL/92 as a warning.

The allowed values of *level* and their meanings are as follows:

- ◆ **e** flag syntax that is not entry-level SQL/92 syntax
- ◆ **i** flag syntax that is not intermediate-level SQL/92 syntax
- ◆ **f** flag syntax that is not full-SQL/92 syntax
- ◆ **w** allow all supported syntax

## CHAPTER 38

# Watcom-SQL Language Reference

### About this chapter

This chapter presents detailed descriptions of the language elements and conventions of Watcom-SQL — one of two SQL dialects native to SQL Anywhere.

### Contents

<b>Topic</b>	<b>Page</b>
Syntax conventions	896
Watcom-SQL language elements	897
Expressions	899
Search conditions	907
Comments in Watcom-SQL	915

☞ For more information about Transact SQL, the other SQL dialect supported by SQL Anywhere, please see "Transact-SQL Procedure Language".

## Syntax conventions

The following conventions are used in the SQL syntax descriptions:

- ◆ **Keywords** All keywords are shown in uppercase. This is often the way SQL statements are typed. However, SQL Anywhere allows all keywords to be in mixed case. Thus `SELECT` is the same as `Select` which is the same as `select`.
- ◆ **Placeholders** Items that the user must replace with appropriate identifiers or expressions are shown in lowercase.
- ◆ **Options** Options are separated by vertical bars. Any one of the items is allowed.
- ◆ **Continuation** Lines beginning with `...` are a continuation of the statements from the previous line.
- ◆ **Lists** Lists are shown with a list element followed by `,...`. This means that one or more list elements are allowed and if more than one is specified, they must be separated by commas.
- ◆ **Optional portions** Optional portions of a statement are enclosed by square brackets. For example, `RELEASE SAVEPOINT [ savepoint-name ]` indicates that the *savepoint-name* is optional. Alternative optional parts of a statement are sometimes listed within the brackets separated by vertical bars. For example, `[ ASC | DESC ]` indicates that `ASC` or `DESC` are optional.
- ◆ **Alternatives** When one of the options must be chosen, the alternatives are enclosed in curly braces. For example `[ QUOTES { ON | OFF } ]` indicates that if the `QUOTES` option is chosen, one of `ON` or `OFF` must be provided. The braces should not be typed.

## Watcom-SQL language elements

The following elements are found in the syntax of many SQL statements. Each of these elements is discussed in more detail later in this chapter.

**column-name** An identifier representing the name of a column.

**condition** An expression that evaluates to TRUE, FALSE, or UNKNOWN. See "Search conditions" on page 907.

**connection-name** An identifier or a string representing the name of an active connection.

**owner** An identifier representing a user ID.

**data-type** A storage data type as described in "SQL Anywhere Data Types".

**expression** An expression, as described in "Expressions" on page 899.

**filename** A string containing a filename.

**host-variable** A C language variable declared as a host variable preceded by a colon.

**identifier** Any string of the characters A through Z, a through z, 0 through 9, underscore ( \_ ), at sign ( @ ), number sign ( # ), or dollar sign ( \$ ). The first character must be a letter. Alternatively, any string of characters can be used as an identifier by enclosing it in quotation marks ("double quotes"). A quotation mark inside the identifier is represented by two quotation marks in a row. Identifiers are truncated to 128 characters. The following are all valid identifiers.

```
Surname
" Surname "
SomeBigName
some_big_name
"Client Number"
"With a quotation " " mark"
```

**indicator-variable** A second host variable of type short int immediately following a normal host variable. It must also be preceded by a colon. Indicator variables are used to pass NULL values to and from the database.

**number** Any sequence of digits followed by an optional decimal part and preceded by an optional negative sign. Optionally, the number can be followed by an E and then an exponent. For example,

```
42
-4.038
.001
3.4e10
1e-10
```

**role-name** An identifier representing the role name of a foreign key.

**search-condition** A condition that evaluates to TRUE, FALSE, or UNKNOWN. See "Search conditions" on page 907.

**string** Any sequence of characters enclosed in apostrophes ('single quotes'). An apostrophe is represented inside the string by two apostrophes in a row. A new line character is represented by a backslash followed by an n (\n). Hexadecimal escape sequences can be used for any character, printable or not. A hexadecimal escape sequence is a backslash followed by an x followed by two hexadecimal digits (for example, \x6d represents the letter m). A backslash character is represented by two backslashes in a row (\). The following are valid strings:

```
'This is a string.'
'John' 's database'
'\x00\x01\x02\x03'
```

**savepoint-name** An identifier representing the name of a savepoint.

**statement-label** An identifier representing the label of a loop or compound statement.

**table-list** A list of table names which may include correlation names. See "FROM clause" in the chapter "Watcom-SQL Statements".

**table-name** An identifier representing the name of a table.

**userid** An identifier representing a user name.

**variable** An identifier representing a variable name.

# Expressions

<b>Syntax</b>	<p>expression:</p> <ul style="list-style-type: none"> <li>  <i>constant</i></li> <li>  [<i>correlation-name</i> .] <i>column-name</i></li> <li>  <i>variable-name</i></li> <li>  <i>function-name</i> ( <i>expression</i>, ... )</li> <li>  - <i>expression</i></li> <li>  <i>expression</i> + <i>expression</i></li> <li>  <i>expression</i> - <i>expression</i></li> <li>  <i>expression</i> * <i>expression</i></li> <li>  <i>expression</i> / <i>expression</i></li> <li>  <i>expression</i> + <i>expression</i></li> <li>  <i>expression</i>    <i>expression</i></li> <li>  ( <i>expression</i> )</li> <li>  ( <i>subquery</i> )</li> <li>  <b>CAST</b> ( <i>expression AS data-type</i> )</li> <li>  <i>if-expression</i></li> </ul> <p><i>constant</i>:</p> <ul style="list-style-type: none"> <li>  <i>integer</i></li> <li>  <i>number</i></li> <li>  '<i>string</i>'</li> <li>  <i>special-constant</i></li> <li>  <i>host-variable</i></li> </ul> <p><i>special-constant</i>:</p> <ul style="list-style-type: none"> <li>  <b>CURRENT DATE</b></li> <li>  <b>CURRENT TIME</b></li> <li>  <b>CURRENT TIMESTAMP</b></li> <li>  <b>NULL</b></li> <li>  <b>SQLCODE</b></li> <li>  <b>SQLSTATE</b></li> <li>  <b>USER</b></li> </ul> <p><i>if-expression</i>:</p> <ul style="list-style-type: none"> <li>  <b>IF</b> <i>condition</i> <b>THEN</b> <i>expression</i> [ <b>ELSE</b> <i>expression</i> ] <b>ENDIF</b></li> </ul>
<b>Usage</b>	Anywhere.
<b>Authorization</b>	Must be connected to the database.
<b>Side effects</b>	None.
<b>See also</b>	Search conditions, on page 907 "SQL Anywhere Data Types"
<b>Description</b>	Expressions are formed from the following elements: <ul style="list-style-type: none"> <li>◆ Constants</li> </ul>

- ◆ Column names
- ◆ Variables
- ◆ Functions
- ◆ Subqueries
- ◆ Operators

## Constants in expressions

Constants are numbers or strings. String constants are enclosed in apostrophes ('single quotes'). An apostrophe is represented inside the string by two apostrophes in a row.

There are several special constants:

- ◆ **CURRENT DATE** The current year, month and day represented in the DATE data type.
- ◆ **CURRENT TIME** The current hour, minute, second and fraction of a second represented in the TIME data type. Although the fraction of a second is stored to 6 decimal places, the current time is limited by the accuracy of the system clock.

Under DOS and Windows, the clock is only accurate to approximately 1/18th of a second rounded to two decimal places. Under QNX, the clock is accurate to the nearest microsecond.

- ◆ **CURRENT TIMESTAMP** Combines CURRENT DATE and CURRENT TIME to form a TIMESTAMP value containing year, month, day, hour, minute, second and fraction of a second. Like CURRENT TIME, the accuracy of the fraction of a second is limited by the system clock.

In embedded SQL, a host variable can also be used in an expression wherever a constant is allowed.

- ◆ **NULL** The NULL value (see "NULL value" in the chapter "Watcom-SQL Statements").
- ◆ **SQLCODE** Current SQLCODE value (see the chapter "SQL Anywhere Database Error Messages").
- ◆ **SQLSTATE** Current SQLSTATE value (see the chapter "SQL Anywhere Database Error Messages").
- ◆ **CURRENT USER** A string containing the user ID of the current connection.



- ◆ **CURRENT PUBLISHER** A string containing publisher user ID of the database for SQL Remote replications.
- ◆ **LAST USER** For INSERTS, this constant has the same effect as CURRENT USER. For UPDATES, if a column with a default value of LAST USER is not explicitly altered, it is changed to the name of the current user. In this way, the LAST USER default indicates the user ID of the user who last modified the row.

When combined with the CURRENT TIMESTAMP, a default value of LAST USER can be used to record (in separate columns) both the user and the date and time a row was last changed..

In Embedded SQL, a host variable can also be used in an expression wherever a constant is allowed.

## Column names in expressions

A column name is an identifier preceded by an optional correlation name. (A correlation name is usually a table name. For more information on correlation names, see "FROM clause" in the chapter "Watcom-SQL Statements".) If a column name has characters other than letters, digits and underscore, it must be surrounded by quotation marks (""). For example, the following are valid column names:

```
employee.name
address
"date hired"
"salary"."date paid"
```

☞ See "Watcom-SQL language elements" on page 897 for a complete description of identifiers.

## Watcom-SQL variables

SQL Anywhere supports three levels of variables:

- ◆ Local variables are defined inside a compound statement in a procedure or batch using the DECLARE statement. They exist only inside the compound statement.

- ◆ Connection-level variables are defined with a **CREATE VARIABLE** statement. They belong to the current connection, and disappear when you disconnect from the database or when you use the **DROP VARIABLE** statement.
- ◆ Global variables are SQL Anywhere-supplied variables that have system-supplied values.

## Local variables

Local variables are declared using the **DECLARE** statement, which can be used only within a compound statement (that is, bracketed by the **BEGIN** and **END** keywords). The variable is initially set as **NULL**. The value of the variable can be set using the **SET** statement, or can be assigned using a **SELECT** statement with an **INTO** clause.

Local variables can be passed as arguments to procedures, as long as the procedure is called from within the compound statement.

The following batch of SQL statements illustrates the use of local variables.

```
BEGIN
    DECLARE local_var INT ;
    SET local_var = 10 ;
    MESSAGE 'local_var = ', local_var ;
END
```

Running this batch from ISQL gives the message `local_var = 10` on the engine window.

The variable `local_var` does not exist outside the compound statement in which it is declared. The following batch is invalid, and gives a column not found error.

```
-- This batch is invalid.
BEGIN
    DECLARE local_var INT ;
    SET local_var = 10 ;
    MESSAGE 'local_var = ', local_var ;
END;
MESSAGE 'local_var = ', local_var ;
```

The following example illustrates the use of **SELECT** with an **INTO** clause to set the value of a local variable:

```
BEGIN
    DECLARE local_var INT ;
    SELECT 10 INTO local_var ;
    MESSAGE 'local_var = ', local_var ;
END
```

Running this batch from ISQL gives the message `local_var = 10` on the engine window.

## Connection-level variables

Connection-level variables are declared with the `CREATE VARIABLE` statement. The `CREATE VARIABLE` statement can be used anywhere except inside compound statements. Connection-level variables can be passed as parameters to procedures.

When a variable is created it is initially set to `NULL`. The value of connection-level variables can be set in the same way as local variables, using the `SET` statement or using a `SELECT` statement with an `INTO` clause. The following batch of SQL statements illustrates the use of connection-level variables.

```
CREATE VARIABLE con_var INT;
SET con_var = 10;
MESSAGE 'con_var = ', con_var;
```

Running this batch from ISQL gives the message `local_var = 10` on the engine window.

Connection-level variables exist until the connection is terminated, or until the variable is explicitly dropped using the `DROP VARIABLE` statement. The following statement drops the variable `con_var`:

```
DROP VARIABLE con_var
```

## Global variables

Global variables are SQL Anywhere-supplied variables that have values set by the SQL Anywhere engine. For example, the global variable `@@version` has a value that is the current version number of the database engine.

Predefined global variables are distinguished from local and connection-level variables by having two `@` signs preceding their names. For example, `@@error`, `@@rowcount` are global variables. Users cannot create global variables, and cannot update the value of global variables directly.

The special constants available in SQL Anywhere, such as `CURRENT DATE`, `CURRENT TIME`, `USER`, `SQLSTATE` and so on are similar to, but not identical to, global variables. The special constants can be used as defaults for columns; global variables cannot.

The following statement retrieves a value of the version global variable.

```
SELECT @@version
```

In procedures and triggers, global variables can be selected into a variable list. The following procedure returns the engine version number in the `ver` parameter.

```
CREATE PROCEDURE VersionProc ( OUT ver
                             NUMERIC ( 5, 2 ) )
BEGIN
    SELECT @@version
    INTO ver;
```

END

In Embedded SQL, global variables can be selected into a host variable list.

The following table lists the global variables available in SQL Anywhere

Variable name	Meaning
<i>@@error</i>	Commonly used to check the error status (succeeded or failed) of the most recently executed statement. It contains 0 if the previous transaction succeeded; otherwise, it contains the last error number generated by the system. A statement such as if @@error != 0 return causes an exit if an error occurs. Every SQL statement resets @error, so the status check must immediately follow the statement whose success is in question.
<i>@@identity</i>	Last value inserted into an IDENTITY column by an insert or select into statement. @@identity is reset each time a row is inserted into a table. If a statement inserts multiple rows, @@identity reflects the IDENTITY value for the last row inserted. If the affected table does not contain an IDENTITY column, @@ identity is set to 0. The value of @@identity is not affected by the failure of an insert or select into statement, or the rollback of the transaction that contained it. @@identity retains the last value inserted into an IDENTITY column, even if the statement that inserted it fails to commit.
<i>@@isolation</i>	Current isolation level. @@isolation takes the value of the active level.
<i>@@procid</i>	Stored procedure ID of the currently executing procedure.
<i>@@rowcount</i>	Number of rows affected by the last statement. @@rowcount is set to zero by any statement which does not return rows, such as an if statement. With cursors, @@rowcount represents the cumulative number of rows returned from the cursor result set to the client, up to the last fetch request.
<i>@@servername</i>	Name of the current database server.
<i>@@sqlstatus</i>	Contains status information resulting from the last fetch statement.
<i>@@version</i>	Version of the current version of SQL Anywhere.

## Functions in expressions

See "Watcom-SQL Functions" for a description of the functions available in SQL Anywhere.

## Subqueries in expressions

A subquery is a SELECT statement enclosed in parentheses. The SELECT statement must contain one and only one select list item. Usually, the subquery is allowed to return only one row. See "Search conditions" on page 907 for other uses of subqueries. A subquery can be used anywhere that a column name can be used. For example, a subquery can be used in the select list of another SELECT statement.

## Watcom-SQL Operators

This section describes the arithmetic and string operators available in SQL Anywhere. For information on comparison operators, see the section "Search conditions" on page 907.

The normal precedence of operations applies. Expressions in parentheses are evaluated first; then multiplication and division before addition and subtraction. String concatenation happens after addition and subtraction.

**expression + expression** Addition. If either expression is the NULL value, the result is the NULL value.

**expression - expression** Subtraction. If either expression is the NULL value, the result is the NULL value.

**- expression** Negation. If the expression is the NULL value, the result is the NULL value.

**expression \* expression** Multiplication. If either expression is the NULL value, the result is the NULL value.

**expression / expression** Division. If either expression is the NULL value or if the second expression is 0, the result is the NULL value.

**expression || expression** String concatenation (two vertical bars). If either string is the NULL value, it is treated as the empty string for concatenation.

**expression + expression** Alternative string concatenation. When using the + concatenation operator, you must ensure the operands are explicitly set to character data types rather than relying on implicit data conversion.

**( expression )** Parentheses.

**IF condition THEN expression1 [ELSE expression2] ENDIF**

Evaluates to the value of *expression1* if the specified search condition is TRUE, the value of *expression2* if *condition* is FALSE, and the NULL value if *condition* is UNKNOWN. (For more information about TRUE, FALSE and UNKNOWN conditions, see "NULL value" in the chapter "Watcom-SQL Statements", and "Search conditions" on page 907.)

## Search conditions

<b>Purpose</b>	To specify a search condition for a WHERE clause, a HAVING clause, a CHECK clause, a JOIN clause or an IF expression.
<b>Syntax</b>	<p>search condition:</p> <pre> <i>expression compare expression</i>   <i>expression compare</i> <b>ANY</b> ( <i>subquery</i> )   <i>expression compare</i> <b>ALL</b> ( <i>subquery</i> )   <i>expression</i> <b>IS</b> [ <b>NOT</b> ] <b>NULL</b>   <i>expression</i> [ <b>NOT</b> ] <b>LIKE</b> <i>expression</i> [<b>ESCAPE</b> <i>expression</i>]   <i>expression</i> [ <b>NOT</b> ] <b>BETWEEN</b> <i>expression</i> <b>AND</b> <i>expression</i>   <i>expression</i> [ <b>NOT</b> ] <b>IN</b> ( <i>expression</i> )   <i>expression</i> [ <b>NOT</b> ] <b>IN</b> ( <i>subquery</i> )   <i>expression</i> [ <b>NOT</b> ] <b>IN</b> ( <i>value-expr1</i> , <i>value-expr2</i> [ , <i>value-expr3</i> ] ... )   <b>EXISTS</b> ( <i>subquery</i> )   <b>NOT</b> <i>condition</i>   <i>condition</i> <b>AND</b> <i>condition</i>   <i>condition</i> <b>OR</b> <i>condition</i>   ( <i>condition</i> )   ( <i>condition</i> , <i>estimate</i> )   <i>condition</i> <b>IS</b> [ <b>NOT</b> ] <b>TRUE</b>   <i>condition</i> <b>IS</b> [ <b>NOT</b> ] <b>FALSE</b>   <i>condition</i> <b>IS</b> [ <b>NOT</b> ] <b>UNKNOWN</b> </pre> <p>compare:</p> <pre> one of = &gt; &lt; &gt;= &lt;= &lt;&gt; != ~= !&lt; !&gt; </pre>
<b>Usage</b>	Anywhere.
<b>Authorization</b>	Must be connected to the database.
<b>Side effects</b>	None.
<b>See also</b>	Expressions, on page 899
<b>Description</b>	<p>Conditions are used as to choose a subset of the rows from a table, or in a control statement such as an IF statement to determine control of flow.</p> <p>SQL conditions do not follow boolean logic, where conditions are either true or false. In SQL, every condition evaluates as one of TRUE, FALSE, or UNKNOWN. This is called three valued logic. The result of a comparison is UNKNOWN if either value being compared is the NULL value. For tables showing how logical operators combine in three-valued logic, see the section "Three-valued logic" on page 914.</p>

Rows satisfy a search condition if and only if the result of the condition is TRUE. Rows for which the condition is UNKNOWN do not satisfy the search condition. For more information about NULL, see "NULL value" in the chapter "Watcom-SQL Statements".

Subqueries form an important class of expression that is used in many search conditions. For information about using subqueries in search conditions, see "Subqueries in search conditions", next.

The different types of search condition are as follows:

- ◆ "Comparison conditions" on page 908.
- ◆ "BETWEEN conditions" on page 909.
- ◆ "LIKE conditions" on page 909.
- ◆ "IN conditions" on page 912.
- ◆ "ALL or ANY conditions" on page 912.
- ◆ "EXISTS conditions" on page 913.
- ◆ "IS NULL conditions" on page 913.
- ◆ "Conditions with logical operators" on page 913.

## Subqueries in search conditions

Subqueries that return exactly one column and either zero or one row can be used in any SQL statement anywhere that a column name could be used, including in the middle of an expression.

For example, expressions can be compared to subqueries in comparison conditions (see "Comparison conditions", next) as long as the subquery does not return more than one row. If the subquery (which must have one column) returns one row, then the value of that row is compared to the expression. If a subquery returns no rows, its value is NULL.

Subqueries that return exactly one column and any number of rows can be used in IN conditions, ANY conditions, and ALL conditions. Subqueries returning any number of columns and rows can be used in EXISTS conditions. These conditions are discussed in the following sections.

## Comparison conditions

The syntax for comparison conditions is as follows:



... *expression compare expression*

where *compare* is a comparison operator. The following comparison operators are available in SQL Anywhere:

operator	description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
!=	Not equal to
<>	Not equal to
!>	Not greater than
!<	Not less than

#### Comparisons are case insensitive

All string comparisons are *case insensitive* unless the database was created as case sensitive.

## BETWEEN conditions

The syntax for BETWEEN conditions is as follows:

... *expr* [ **NOT** ] **BETWEEN** *start-expr* **AND** *end-expr*

The BETWEEN condition can evaluate as TRUE, FALSE, or UNKNOWN. Without the NOT keyword, the condition evaluates as TRUE if *expr* is between *start-expr* and *end-expr*. The NOT keyword reverses the meaning of the condition, leaving UNKNOWN unchanged.

The BETWEEN conditions is equivalent to a combination of two inequalities:

*expr* >= *start-expr* **AND** *expr* <= *end-expr*

## LIKE conditions

The syntax for LIKE conditions is as follows:

... *expression* [ **NOT** ] **LIKE** *pattern* [ **ESCAPE** *escape-expr* ]

The LIKE condition can evaluate as TRUE, FALSE, or UNKNOWN.

Without the NOT keyword, the condition evaluates as TRUE if *expression* matches the *pattern*. If either *expression* or *pattern* is the NULL value, this condition is UNKNOWN. The NOT keyword reverses the meaning of the condition, leaving UNKNOWN unchanged.

The pattern may contain any number of wild cards. The wild cards are:

Wild card	Matches
_ (underscore)	Any one character
% (percent)	Any string of zero or more characters
[]	Any single character in the specified range or set
[^]	Any single character not in the specified range or set

All other characters must match exactly.

For example, the search condition

```
... name LIKE 'a%b_'
```

is TRUE for any row where name starts with the letter a and has the letter b as its second last character.

If an *escape-expr* is specified, it must evaluate to a single character. The character can precede a percent, an underscore, a left square bracket, or another escape character in the *pattern* to prevent the special character from having its special meaning. When escaped in this manner, a percent will match a percent, and an underscore will match an underscore.

All patterns of length 126 characters or less are supported. Patterns of length greater than 254 characters are not supported. Some patterns of length between 127 and 254 characters are supported, depending on the contents of the pattern.

#### Using ranges and sets in patterns

Ranges and sets of characters can be given in LIKE search conditions using square brackets.

#### Searching for one of a set of characters

A set of characters to look for is specified by listing the characters inside the brackets. For example, the following condition finds the strings *smith* and *smyth*:

```
... LIKE 'sm[iy]th'
```

#### Searching for one of a range of characters

A range of characters to look for is specified by giving the ends of the range, separated by a hyphen. For example, the following condition finds the strings *bough* and *rough*, but not *tough*:

```
... LIKE '[a-r]ough'
```

The range of characters [a-z] is interpreted as "greater than or equal to a, and less than or equal to z", where the greater than and less than operations are carried out within the collation of the database. For information on ordering of characters within a collation, see the chapter "Database Collations".

The lower end of the range must precede the higher end of the range. For example, any LIKE condition containing the expression [z-a] returns no rows, as no character matches the [z-a] range.

Unless the database was created as a case-insensitive database, the range of characters is case insensitive. For example, the following condition finds the strings *Bough*, *rough*, and *TOUGH*:

```
... LIKE '[a-z]ough'
```

If the database is created as a case-sensitive database, the search condition is case sensitive also.

#### Combining searches for ranges and sets

You can combine ranges and sets within a square bracket. For example, the following condition finds the strings *bough*, *rough*, and *tough*:

```
... LIKE '[a-rt]ough'
```

The bracket [*a-mpqs-z*] is interpreted as "exactly one character that is either in the range *a* to *m* inclusive, or is *p*, or is *q*, or is in the range *s* to *z* inclusive".

#### Searching for one character not in a range

The caret character (^) is used to specify a range of characters that is excluded from a search. For example, the following condition finds the string *tough*, but not the strings *rough*, or *bough*:

```
... LIKE '[^a-r]ough'
```

The caret negates the entire rest of the contents of the brackets. For example, the bracket [*^a-mpqs-z*] is interpreted as "exactly one character that is not in the range *a* to *m* inclusive, is not *p*, is not *q*, and is not in the range *s* to *z* inclusive".

#### Special cases of ranges and sets

Any single character in square brackets means that character. For example, [*a*] matches just the character *a*. [*^*] matches just the caret character, [*%*] matches just the percent character (the percent character does not act as a wild card in this context), and [*\_*] matches just the underscore character. Also, [*[]*] matches just the character [*.*].

Other special cases are as follows:

- ◆ The expression [*a-*] matches either of the characters *a* or *-*.
- ◆ The expression [*[]*] is never matched and always returns no rows.

- ◆ The expressions `[` or `[abp-q` are ill-formed expressions, and give syntax errors.
- ◆ You cannot use wild cards inside square brackets. The expression `[a%b]` finds one of `a`, `%`, or `b`.
- ◆ You cannot use the caret character to negate ranges except as the first character in the bracket. The expression `[a^b]` finds one of `a`, `^`, or `b`.

## IN conditions

The syntax for IN conditions is as follows:

```
...expression [ NOT ] IN ( subquery )
| expression [ NOT ] IN ( expression )
| expression [ NOT ] IN (value-expr1 , value-expr2 [, value-expr3] ...
| )
```

Without the NOT keyword, the IN conditions is TRUE if *expression* equals any of the listed values, UNKNOWN if *expression* is the NULL value, and FALSE otherwise. The NOT keyword reverses the meaning of the condition, leaving UNKNOWN unchanged.

## ALL or ANY conditions

The syntax for ANY conditions is

```
... expression compare ANY ( subquery )
```

where *compare* is a comparison operator.

For example, an ANY condition with an equality operator:

```
... expression = ANY ( subquery )
```

is TRUE if *expression* is equal to any of the values in the result of the subquery, and FALSE if the expression is not NULL and does not equal any of the columns of the subquery. The ANY condition is UNKNOWN if *expression* is the NULL value unless the result of the subquery has no rows, in which case the condition is always FALSE.

The keyword SOME can be used instead of ANY.

## EXISTS conditions

The syntax for EXISTS conditions is as follows:

... **EXISTS**( *subquery* )

The EXISTS condition is TRUE if the subquery result contains at least one row, and FALSE if the subquery result does not contain any rows. The EXISTS condition cannot be UNKNOWN.

## IS NULL conditions

The syntax for IS NULL conditions is as follows:

*expression* **IS** [ **NOT** ] **NULL**

Without the NOT keyword, the IS NULL condition is TRUE if the expression is the NULL value, and as FALSE otherwise. The NOT keyword reverses the meaning of the condition.

## Conditions with logical operators

Search conditions can be combined using AND, OR and NOT.

Conditions are combined using AND as follows:

... *condition1* **AND** *condition2*

The combined condition is TRUE if both conditions are TRUE, FALSE if either condition is FALSE, and UNKNOWN otherwise.

Conditions are combined using OR as follows:

... *condition1* **OR** *condition2*

The combined condition is TRUE if either condition is TRUE, FALSE if both conditions are FALSE, and UNKNOWN otherwise.

The result of a comparison is UNKNOWN if either value being compared is the NULL value. Rows satisfy a search condition if and only if the result of the condition is TRUE.

## NOT conditions

The syntax for NOT conditions is as follows:

... **NOT** *condition1*

The NOT condition is TRUE if condition1 is FALSE, FALSE if condition1 is TRUE and UNKNOWN if condition1 is UNKNOWN.

## Truth value conditions

The syntax for truth value conditions is as follows:

... **IS** [ **NOT** ] *truth-value*

Without the NOT keyword, the condition is TRUE if the *condition* evaluates to the supplied *truth-value*, which must be one of TRUE, FALSE, or UNKNOWN. Otherwise, the value is FALSE. The NOT keyword reverses the meaning of the condition, leaving UNKNOWN unchanged.

## Three-valued logic

The following tables show how the AND, OR, NOT, and IS logical operators of SQL work in three-valued logic.

<b>AND</b>	<b>TRUE</b>	<b>FALSE</b>	<b>UNKNOWN</b>
<b>TRUE</b>	TRUE	FALSE	UNKNOWN
<b>FALSE</b>	FALSE	FALSE	FALSE
<b>UNKNOWN</b>	UNKNOWN	FALSE	UNKNOWN

<b>OR</b>	<b>TRUE</b>	<b>FALSE</b>	<b>UNKNOWN</b>
<b>TRUE</b>	TRUE	TRUE	TRUE
<b>FALSE</b>	TRUE	FALSE	UNKNOWN
<b>UNKNOWN</b>	TRUE	UNKNOWN	UNKNOWN

<b>NOT</b>	<b>TRUE</b>	<b>FALSE</b>	<b>UNKNOWN</b>
	FALSE	TRUE	UNKNOWN

<b>IS</b>	<b>TRUE</b>	<b>FALSE</b>	<b>UNKNOWN</b>
<b>TRUE</b>	TRUE	FALSE	FALSE
<b>FALSE</b>	FALSE	TRUE	FALSE
<b>UNKNOWN</b>	FALSE	FALSE	TRUE

## Comments in Watcom-SQL

Comments are used to attach explanatory text to SQL statements or statement blocks. Comments are not executed by the database engine.

Several comment indicators are available in SQL Anywhere.

**-- (Double hyphen.)** Any remaining characters on the line are ignored by the database engine. This is the SQL/92 comment indicator.

**% (Percent sign.)** The percent sign has the same meaning as the double hyphen.

**// (Double slash.)** The double slash has the same meaning as the double hyphen.

**/\* ... \*/ (Slash-asterisk.)** Any characters between the two comment markers are ignored. The two comment markers may be on the same or different lines. Comments indicated in this style can be nested.

### Transact-SQL compatibility

The double-hyphen and the slash-asterisk comment styles are compatible with Transact-SQL.

The following examples illustrate the use of comments:

```
CREATE FUNCTION fullname (firstname CHAR(30),
  lastname CHAR(30))
  RETURNS CHAR(61)
  -- fullname concatenates the firstname and lastname
  -- arguments with a single space between.
  BEGIN
    DECLARE name CHAR(61);
    SET name = firstname || ' ' || lastname;
    RETURN ( name );
  END

/*
  Lists the names and employee IDs of employees
  who work in the sales department.
*/
CREATE VIEW SalesEmployee AS
  SELECT emp_id, emp_lname, emp_fname
  FROM "dba".employee
  WHERE dept_id = 200
```





# SQL Anywhere Data Types

## About this chapter

This chapter describes the data types supported by SQL Anywhere.

## Contents

Topic	Page
Character data types	918
Numeric data types	920
Date and time data types	922
Binary data types	926
User-defined data types	927
Data type conversions	929

## For more information

- ◆ The different kinds of data type are discussed in the following sections.
- ◆ Data type conversions are described in the section "Data type conversions" on page 929.
- ◆ Some of the data types listed have Transact-SQL equivalents. These equivalents are not listed here. For a description of Transact-SQL data types supported by SQL Anywhere, see the chapter "Using Transact-SQL with SQL Anywhere".

# Character data types

**Purpose** For storing strings of letters, numbers and symbols.

**Syntax** Character data types:  
CHAR [ ( *max-length* ) ]  
| CHARACTER [ ( *max-length* ) ]  
| CHARACTER VARYING [ ( *max-length* ) ]  
| LONG VARCHAR  
| VARCHAR [ ( *max-length* ) ]

**Description** **CHAR [(max-length)]** Character data of maximum length *max-length*. If *max-length* is omitted, the default is 1. The maximum size allowed is 32,767.

☞ See the notes below on character data representation in the database, and on storage of long strings.

**CHARACTER [(max-length)]** Same as CHAR[(max-length)].

**CHARACTER VARYING[(max-length)]** Same as CHAR[(max-length)].

**LONG VARCHAR** Arbitrary length character data. The maximum size is limited by the maximum size of the database file (currently 2 gigabytes).

**VARCHAR [(max-length)]** Same as CHAR[(max-length)].

**Notes** Character data is placed in the database using the exact binary representation that is passed from the application. This usually means that character data is stored in the database with the binary representation of the current **code page**. The code page is the character set representation used by IBM-compatible personal computers. You can find documentation about code pages in the documentation for your operating system.

All code pages are the same for the first 128 characters. If you use special characters from the top half of the code page (accented international language characters), you must be careful with your databases. In particular, if you copy the database to a different machine using a different code page, those special characters will be retrieved from the database using the original code page representation. With the new code page, they will appear on the screen to be the wrong characters.

This problem also appears if you have two clients using the same multi-user server, but running with different code pages. Data inserted or updated by one client may appear incorrect to another.

This problem is quite complex. If any of your applications use the extended characters in the upper half of the code page, make sure that all clients and all machines using the database use the same or a compatible code page.

### Long strings

SQL Anywhere treats CHAR, VARCHAR, and LONG VARCHAR columns all as the same type. Values up to 254 characters are stored as short strings, which are stored with a preceding length byte. Any values that are longer than 255 bytes are considered long strings. Characters after the 255th are stored separate from the row containing the long string value.

There are several functions (see "Watcom-SQL Functions") that will ignore the part of any string past the 255th character. They are soundex, similar, and all of the date functions. Also, any arithmetic involving the conversion of a long string to a number will work on only the first 255 characters. It would be extremely unusual to run in to one of these limitations.

All other functions and all other operators will work with the full length of long strings.

## Numeric data types

**Purpose** For storing numerical data.

**Syntax** Numeric data types  
**DECIMAL** [ ( *precision* [ , *scale* ] ) ]  
| **DOUBLE**  
| **FLOAT** [ ( *precision* ) ]  
| **INT**  
| **INTEGER**  
| **NUMERIC** [ ( *precision* [ , *scale* ] ) ]  
| **REAL**  
| **SMALLINT**  
| **TINYINT**

**Description** **DECIMAL [(precision[,scale])]** A decimal number with *precision* total digits and with *scale* of the digits after the decimal point. The defaults are *scale* = 6 and *precision* = 30.

These defaults can be changed with the SET statement (see "SET OPTION statement" in the chapter "Watcom-SQL Statements").

The storage required for a decimal number can be computed as:

$$2 + \text{int}((\text{before}+1) / 2) + \text{int}((\text{after}+1)/2)$$

where *int* takes the integer portion of its argument, and *before* and *after* are the number of significant digits before and after the decimal point. Note that the storage is based on the value being stored, not on the maximum precision and scale allowed in the column.

**DOUBLE** A double precision floating-point number stored in 8 bytes. The range of values is 2.22507385850720160e-308 to 1.79769313486231560e+308. Values held as **DOUBLE** are accurate to 15 significant digits, but may be subject to round-off error beyond the fifteenth digit.

**FLOAT [ ( precision ) ]** If *precision* is not supplied, the **FLOAT** data type is the same as the **REAL** data type. If *precision* supplied, then the **FLOAT** data type is the same as the **REAL** or **DOUBLE** data type, depending on the value of the precision. The cutoff between **REAL** and **DOUBLE** is platform dependent.

When a column is created using the **FLOAT(precision)** data type, columns on all platforms are guaranteed to hold the values to at least the specified minimum precision. In contrast, **REAL** and **DOUBLE** do not guarantee a platform-independent minimum precision.

**INT** Signed integer of maximum value 2,147,483,647 requiring 4 bytes of storage.

**INTEGER** Same as INT.

**NUMERIC [(precision[,scale])]** Same as DECIMAL.

**REAL** A single precision floating-point number stored in 4 bytes. The range of values is 1.175494351e-38 to 3.402823466e+38. Values held as REAL are accurate to 6 significant digits, but may be subject to round-off error beyond the sixth digit.

**SMALLINT** Signed integer of maximum value 32,767 requiring 2 bytes of storage.

**TINYINT** Signed integer of maximum value 255 requiring 2 bytes or 4 bytes of storage.

#### Notes

- ◆ The INTEGER, NUMERIC and DECIMAL data types are sometimes called *exact* numeric data types, in contrast to the *approximate* numeric data types FLOAT, DOUBLE, and REAL. The exact numeric data types are those for which precision and scale values can be specified, while approximate numeric data types are stored in a predefined manner. Columns holding exact numeric data are accurate after arithmetic operations to the least significant digit specified.
- ◆ In SQL Anywhere TINYINT columns should not be fetched into Embedded SQL variables defined as char or unsigned char, since the result is an attempt to convert the value of the column to a string and then assign the first byte to the variable in the program.
- ◆ Before release 5.5, hexadecimal constants greater than four bytes were treated as string constants, and others were treated as integers. The new default behavior is to treat them as binary type constants. To use the historical behavior, set the TSQL\_HEX\_CONSTANTS database option to OFF.

## Date and time data types

**Purpose** For storing dates and times.

**Syntax** Date and time data types  
**DATE**  
| **TIME**  
| **TIMESTAMP**

**Description**

**DATE** A calendar date, such as a year, month and day. The year can be from the year 0001 to 9999. For historical reasons, a DATE column can also contain an hour and minute, but the **TIMESTAMP** data type is now recommended for anything with hours and minutes. A DATE value requires 4 bytes of storage.

**TIME** time of day, containing hour, minute, second and fraction of a second. The fraction is stored to 6 decimal places. A TIME value requires 8 bytes of storage. (ODBC standards restrict TIME data type to an accuracy of seconds. For this reason you should not use TIME data types in WHERE clause comparisons that rely on a higher accuracy than seconds.)

**TIMESTAMP** point in time, containing year, month, day, hour, minute, second and fraction of a second. The fraction is stored to 6 decimal places. A **TIMESTAMP** value requires 8 bytes of storage.

Although the range of possible dates for the **TIMESTAMP** data type is the same as the DATE type, 0001 to 9999, the useful range of **TIMESTAMP** date types is from 1600-02-28 23:59:59 to 7911-01-01 00:00:00. Prior to, and after this range the time portion of the **TIMESTAMP** may be incomplete.

## Sending dates and times to the database

Dates and times may be sent to the database in one of the following ways:

- ◆ Using any interface, as a string
- ◆ Using ODBC, as a **TIMESTAMP** structure
- ◆ Using Embedded SQL, as a **SQLDATETIME** structure

When a time is sent to the database as a string (for the **TIME** data type) or as part of a string (for **TIMESTAMP** or **DATE** data types), the hours, minutes, and seconds must be separated by colons in the format *hh:mm:ss.sss*, but can appear anywhere in the string. The following are valid and unambiguous strings for specifying times:

```

21:35 -- 24 hour clock if no am or pm specified
10:00pm -- pm specified, so interpreted as 12 hour
clock
10:00 -- 10:00am in the absence of pm
10:23:32.234 -- seconds and fractions of a second
included

```

When a date is sent to the database as a string, conversion to a date is automatic. The string can be supplied in one of two ways:

- ◆ As a string of format *yyyy/mm/dd* or *yyyy-mm-dd*, which is interpreted unambiguously by the database
- ◆ As a string interpreted according to the `DATE_ORDER` database option

## Unambiguous dates and times

Dates in the format *yyyy/mm/dd* or *yyyy-mm-dd* are always recognized unambiguously as dates regardless of the `DATE_ORDER` setting. Other characters can be used as separators instead of "/" or "-"; for example, "?", a space character, or ",". You should use this format in any context where different users may be employing different `DATE_ORDER` settings. For example, in stored procedures, use of the unambiguous date format prevents misinterpretation of dates according to the user's `DATE_ORDER` setting.

Also, a string of the form *hh:mm:ss.sss* is interpreted unambiguously as a time.

For combinations of dates and times, any unambiguous date and any unambiguous time yield an unambiguous date-time value. Also, the form

```
YYYY-MM-DD HH.MM.SS.SSSSS
```

is an unambiguous date-time value. Periods can be used in the time only in combination with a date.

In other contexts, a more flexible date format can be used. SQL Anywhere can interpret a wide range of strings as formats. The interpretation depends on the setting of the database option `DATE_ORDER`. The `DATE_ORDER` database option can have the value 'MDY', 'YMD', or 'DMY' (see "SET OPTION statement" in the chapter "Watcom-SQL Statements"). For example, the following statement sets the `DATE_ORDER` option to 'DMY':

```
SET OPTION DATE_ORDER = 'DMY' ;
```

The default DATE\_ORDER setting is 'YMD'. The SQL Anywhere ODBC driver sets the DATE\_ORDER option to 'YMD' whenever a connection is made. The value can still be changed using the SET OPTION statement.

The database option DATE\_ORDER determines whether the string 10/11/12 is interpreted by the database as Oct 11 1912, Nov 12 1910, or Nov 10 1912. The year, month, and day of a date string should be separated by some character (for example /, -, or space) and appear in the order specified by the DATE\_ORDER option. The year can be supplied as either 2 or 4 digits, with 2 digit years defaulting to the 20th century. The month can be the name or number of the month. The hours and minutes are separated by a colon, but can appear anywhere in the string.

With an appropriate setting of DATE\_ORDER, the following strings are all valid dates:

```
92-05-23 21:35
92/5/23
1992/05/23
May 23 1992
23-May-1992
Tuesday May 23, 1992 10:00pm
```

If a string contains only a partial date specification, default values are used to fill out the date. The following defaults are used:

**year** This year

**month** No default

**day** 1 (useful for month fields; for example, 'May 1992' will be the date '1992-05-01 00:00')

**hour, minute, second, fraction** 0

## Retrieving dates and times from the database

Dates and times may be retrieved from the database in one of the following ways:

- ◆ Using any interface, as a string
- ◆ Using ODBC, as a TIMESTAMP structure
- ◆ Using embedded SQL, as a SQLDATETIME structure



When a date or time is retrieved as a string, it is retrieved in the format specified by the database options `DATE_FORMAT`, `TIME_FORMAT` and `TIMESTAMP_FORMAT`. For descriptions of these options, see "SET OPTION statement" in the chapter "Watcom-SQL Statements".

☞ For information on functions dealing with dates and times, see "Date and time functions" in the chapter "Watcom-SQL Functions". The following arithmetic operators are allowed on dates:

- ◆ **timestamp + integer** Add the specified number of days to a date or timestamp.
- ◆ **timestamp - integer** Subtract the specified number of days from a date or timestamp.
- ◆ **date - date** Compute the number of days between two dates or timestamps.
- ◆ **date + time** Create a timestamp combining the given date and time.

## Date and time comparisons

The `DATE` data type also contains a time. If the time is not specified when a date is entered into the database, the time defaults to 0:00 or 12:00am (midnight). Any date comparisons always involve the times as well. A database date value of '1992-05-23 10:00' will not be equal to the constant '1992-05-23'. The `DATEFORMAT` function or one of the other date functions can be used to compare parts of a date and time field. For example:

```
DATEFORMAT(invoice_date, 'yyyy/mm/dd') = '1992/05/23'
```

If a database column requires only a date, client applications should ensure that times are not specified when data is entered into the database. This way, comparisons with date-only strings will work as expected.

If you wish to compare a date to a string *as a string*, you must use the `DATEFORMAT` function or `CAST` function to convert the date to a string before comparing.

## Binary data types

<b>Purpose</b>	For storing binary data, including images and other information that is not interpreted by the database.
<b>Syntax</b>	Binary data types: <b>BINARY</b> [ ( <i>max-length</i> ) ] <b>LONG BINARY</b>
<b>Description</b>	<b>BINARY [(max-length)]</b> Binary data of maximum length <b>size</b> (in bytes). If <b>size</b> is omitted, the default is 1. The maximum size allowed is 32,767. The <b>BINARY</b> data type is identical to the <b>CHAR</b> data type except when used in comparisons. <b>BINARY</b> values will be compared exactly while <b>CHAR</b> values are compared without respect to upper/lowercase (depending on the case-sensitivity of the database) or accented characters.  <b>LONG BINARY</b> Arbitrary length binary data. The maximum size is limited by the maximum size of the database file (currently 2 gigabytes).

## User-defined data types

### Purpose

User-defined data types are aliases for built-in data types, including precision and scale values where applicable, and optionally including DEFAULT values and CHECK conditions.

User-defined data types, also called **domains**, allow columns throughout a database to be automatically defined on the same data type, with the same NULL or NOT NULL condition, with the same DEFAULT setting, and with the same CHECK condition. This encourages consistency throughout the database.

### Simple user-defined data types

User-defined data types are created using the CREATE DATATYPE statement. For full description of the syntax, see "CREATE DATATYPE statement" in the chapter "Watcom-SQL Statements".

The following statement creates a data type named street\_address, which is a 35-character string.

```
CREATE DATATYPE street_address CHAR( 35 )
```

CREATE DOMAIN can be used as an alternative to CREATE DATATYPE, and is recommended, as CREATE DOMAIN is the syntax used in the draft SQL/3 standard.

Resource authority is required to create data types. Once a data type is created, the user ID that executed the CREATE DATATYPE statement is the owner of that data type. Any user can use the data type, and unlike other database objects, the owner name is never used to prefix the data type name.

The street\_address data type may be used in exactly the same way as any other data type when defining columns. For example, the following table with two columns has the second column as a street\_address column:

```
CREATE TABLE twocol (
  id INT,
  street street_address
)
```

User-defined data types can be dropped by their owner or by the DBA using the DROP DATATYPE statement:

```
DROP DATATYPE street_address
```

This statement can be carried out only if the data type is not used in any table in the database.

### Constraints and defaults with user-defined data types

Many of the attributes associated with columns, such as allowing NULL values, having a DEFAULT value, and so on, can be built into a user-defined data type. Any column that is defined on the data type automatically inherits the NULL setting, CHECK condition, and DEFAULT values. This allows uniformity to be built into columns with a similar meaning throughout a database.

For example, many primary key columns in the sample database are integer columns holding ID numbers. The following statement creates a data type that may be useful for such columns:

```
CREATE DATATYPE id INT
NOT NULL
DEFAULT AUTOINCREMENT
CHECK( @col > 0 )
```

Any column created using the data type `id` is not allowed to hold NULLs, defaults to an autoincremented value, and must hold a positive number. Any identifier could be used instead of `col` in the `@col` variable.

The attributes of the data type can be overridden if needed by explicitly providing attributes for the column. A column created on data type `id` with NULL values explicitly allowed does allow NULLs, regardless of the setting in the `id` data type.

## Data type conversions

Type conversions happen automatically, or they can be explicitly requested using the CAST or CONVERT function.

If a string is used in a numeric expression or as an argument to a function expecting a numeric argument, the string is converted to a number before use.

If a number is used in a string expression or as a string function argument, then the number is converted to a string before use.

All date constants are specified as strings. The string is automatically converted to a date before use.

There are certain cases where the automatic database conversions are not appropriate.

```
'12/31/90' + 5 -- SQL Anywhere tries to convert the
string to a number
'a' > 0 -- SQL Anywhere tries to convert 'a' to a
number
```

The CAST or CONVERT functions can be used to force type conversions. For information about the CAST and CONVERT functions, see "Data type conversion functions" in the chapter "Watcom-SQL Functions".

The following functions can also be used to force type conversions (see "Watcom-SQL Functions").

**DATE( value )** Converts the expression into a date, and removes any hours, minutes or seconds. Conversion errors may be reported.

**STRING( value )** similar to CAST( value AS CHAR ), except that string( NULL ) is the empty string ("), while CAST( NULL AS CHAR ) is the NULL value.

**VALUE+0.0** Equivalent to CAST( value AS DECIMAL ).

## Year 2000 compliance

The problem of handling dates, in particular year values beyond the year 2000, is a significant issue for the computer industry.

This section examines the year 2000 compliance of SQL Anywhere. It illustrates how date values are handled internally by SQL Anywhere, and how SQL Anywhere handles ambiguous date information, such as the conversion of a two digit year string value.

Users of Sybase SQL Anywhere and its predecessors can be assured that dates are handled and stored internally in a manner not adversely effected by the transition from the 20th century to the 21st century.

Consider the following measurements of SQL Anywhere's year 2000 compliance:

- ◆ SQL Anywhere always returns correct values for any legal arithmetic and logical operations on dates, regardless of whether the calculated values span different centuries.
- ◆ At all times SQL Anywhere's internal storage of dates explicitly includes the century portion of a year value.
- ◆ SQL Anywhere's operation is unaffected by any return value, including the current date.
- ◆ Date values can always be outputted in full century format.

Many of the date-related topics summarized in this section are explained in greater detail in other parts of the documentation.

### How dates are stored

Dates containing year values are used internally and stored in SQL Anywhere databases using either of the following data types:

<b>Data type</b>	<b>Contains</b>	<b>Stored in</b>	<b>Range of possible values</b>
DATE	Calendar date (year, month, day)	4-bytes	0001-01-01 to 9999-12-31

Data type	Contains	Stored in	Range of possible values
TIMESTAMP	Time stamp (year, month, day, hour minute, second, and fraction of second accurate to 6 decimal places.)	8-bytes	0001-01-01 to 9999-12-31 (precision of time portion of TIMESTAMP is dropped prior to 1600-02-28 23:59:59 and after 7911-01-01 00:00:00)

☞ For more information on SQL Anywhere date and time data types see "Date and time data types", on page 922.

## Sending and retrieving date values

Date values are stored within SQL Anywhere as either a DATE or TIMESTAMP data type, but they are passed to and retrieved from SQL Anywhere using either of three methods:

- ◆ As a string, using any SQL Anywhere programming interface.
- ◆ As a TIMESTAMP structure using ODBC.
- ◆ As a SQLDATETIME structure using Embedded SQL.

A string containing a date value is considered unambiguous and is automatically converted to a DATE or TIMESTAMP data type without potential for misinterpretation if it is passed using the following format: *yyyy-mm-dd* (the "-" dash separator is one of several characters that are permitted).

Date formats other than *yyyy-mm-dd* can be used by setting the DATE\_FORMAT database option (see "SET OPTION statement" in the chapter "Watcom-SQL Statements").

☞ For more information on unambiguous date formats, see "Unambiguous dates and times", on page 923.

☞ For more information on the ODBC TIMESTAMP structure see the Microsoft Open Database Connectivity SDK, or "Sending dates and times to the database", page 922.

Used in the development of C programs, an embedded SQL SQLDATETIME structure's year value is a 16-bit signed integer.

☞ For more information on the SQLDATETIME data type, see "Embedded SQL interface data types" in the chapter "The Embedded SQL Interface".

## Leap years

The year 2000 is also a leap year, with an additional day in the month of February. SQL Anywhere uses a globally accepted algorithm for determining which years are leap years. Using this algorithm, a year is considered a leap year if it is divisible by four, unless the year is a century date (such as the year 1900), in which case it is a leap year if it is divisible by 400.

SQL Anywhere handles all leap years correctly. For example:

The following SQL statement results in a return value of "Tuesday":

```
SELECT DAYNAME('2000-02-29');
```

SQL Anywhere accepts Feb 29, 2000 — a leap year — as a date and using this date determines the day of the week on which that date occurs.

However, the following statement is rejected by SQL Anywhere:

```
SELECT DAYNAME('2001-02-29');
```

This statement results in an error (cannot convert '2001-02-29' to a date) because Feb 29 does not exist in the year 2001, which it does not.

## Ambiguous string to date conversions

SQL Anywhere automatically converts a string into a date when a date value is expected, even if the year is represented in the string by only two digits.

If the century portion of a year value is omitted, SQL Anywhere's method of conversion is determined by the NEAREST\_CENTURY database option.

The NEAREST\_CENTURY database option is a numeric value that acts as a break point between 1900 date values and 2000 date values.

Two digit years less than the NEAREST\_CENTURY value are converted to 20yy, while years greater than or equal to the value are converted to 19yy.



**Ambiguous date conversion example**

If this option is not set, the default setting of 0 is assumed, thus adding 1900 to two digit year strings and placing them in the 20th century.

This `NEAREST_CENTURY` option was introduced in SQL Anywhere Release 5.5.

The following statement creates a table that can be used to illustrate the conversion of ambiguous date information in SQL Anywhere.

```
CREATE TABLE T1 (C1 DATE);
```

The table T1 contains one column, C1, of the type DATE.

The following statement inserts a date value into the column C1. SQL Anywhere automatically converts a string that contains an ambiguous year value, one with two digits representing the year but nothing to indicate the century.

```
INSERT INTO T1 VALUES('00-01-01');
```

By default, the `NEAREST_CENTURY` option is set to 0, thus SQL Anywhere converts the string into the date 1900-01-01. The following statement verifies the result of this insert.

```
SELECT * FROM T1;
```

Changing the `NEAREST_CENTURY` option using the following statement alters the conversion process.

```
SET OPTION NEAREST_CENTURY = 25;
```

When `NEAREST_CENTURY` option is set to 25, executing the previous insert using the same statement will create a different date value:

```
INSERT INTO T1 VALUES('00-01-01');
```

The above statement now results in the insertion of the date 2000-01-01. Use the following statement to verify the results.

```
SELECT * FROM T1;
```

**Date to string conversions**

SQL Anywhere provides several functions for converting SQL Anywhere date and time values into a wide variety of strings and other expressions. It is possible in converting a date value into a string to reduce the year portion into a two digit number representing the year, thereby losing the century portion of the date.

**Wrong century values**

Consider the following statement, which incorrectly converts a string representing the date Jan 1, 2000 into a string representing the date Jan 1, 1900 even though no database error occurs.

```
SELECT DATEFORMAT (
    DATEFORMAT('2000-01-01', 'Mmm dd/yy' ),
    'yyyy-Mmm-dd' )
AS Wrong_year;
```

Although the unambiguous date string 2000-01-01 is automatically and correctly converted by SQL Anywhere into a date value, the 'Mmm dd/yy' formatting of the inner, or nested DATEFORMAT function drops the century portion of the date when it is converted back to a string and passed to the outer DATEFORMAT function.

Because the database option NEAREST\_CENTURY, in this case, is set to 0 the outer DATEFORMAT function converts the string representing a date with a two digit year value into a year in the 20th century.

☞ For more information about ambiguous string conversions, see "Ambiguous string to date conversions", on page 932.

☞ For more information on date and time functions, see "Date and time functions" in the chapter "Watcom-SQL Functions".

## CHAPTER 40

# Watcom-SQL Functions

### About this chapter

This chapter describes the built-in functions supported by SQL Anywhere. Functions are used to return information from the database. They are allowed anywhere an expression is allowed.

#### **NULL value**

Unless otherwise stated, any function that receives the NULL value as a parameter returns a NULL value.

### Contents

<b>Topic</b>	<b>Page</b>
Aggregate functions	936
Numeric functions	938
String functions	941
Date and time functions	945
Data type conversion functions	951
System functions	953
Miscellaneous functions	965

### For more information

Some of the functions listed have Transact-SQL equivalents that are not listed here. For a description of Transact-SQL functions supported by SQL Anywhere, see the chapter "Using Transact-SQL with SQL Anywhere".

# Aggregate functions

<b>Purpose</b>	Aggregate functions summarize data over a group of rows from the database.
<b>Syntax</b>	Aggregate function: <b>AVG</b> ( <i>aggregate-parm</i> )   <b>COUNT</b> ( * )   <b>COUNT</b> ( <i>aggregate-parm</i> )   <b>LIST</b> ( <i>aggregate-parm</i> )   <b>MAX</b> ( <i>aggregate-parm</i> )   <b>MIN</b> ( <i>aggregate-parm</i> )   <b>SUM</b> ( <i>aggregate-parm</i> )  aggregate-parm: <b>DISTINCT</b> <i>column-name</i>   <i>expression</i>
<b>See also</b>	Numeric functions String functions Date and time functions Data type conversion functions System functions Miscellaneous functions
<b>Description</b>	<p>Aggregate functions summarize data over a group of rows from the database. The groups are formed using the <b>GROUP BY</b> clause of the <b>SELECT</b> statement. Aggregate functions are only allowed in the select list and in the <b>HAVING</b> and <b>ORDER BY</b> clauses of a <b>SELECT</b> statement.</p> <p><b>AVG( numeric-expr )</b> Computes the average of <i>numeric-expr</i> for each group of rows. This average does not include rows where the <i>expression</i> is the <b>NULL</b> value. Returns the <b>NULL</b> value for a group containing no rows.</p> <p><b>AVG( DISTINCT column-name )</b> Computes the average of the unique values in <i>column-name</i>. This is of limited usefulness, but is included for completeness.</p> <p><b>COUNT( * )</b> Returns the number of rows in each group.</p> <p><b>COUNT( expression )</b> Returns the number of rows in each group where the <i>expression</i> is not the <b>NULL</b> value.</p> <p><b>COUNT( DISTINCT column-name )</b> Returns the number of different values in the column with name <i>column-name</i>. Rows where the value is the <b>NULL</b> value are not included in the count.</p>

**LIST( string-expr )** Returns a string containing a comma-separated list composed of all the values for *string-expr* in each group of rows. Rows where *string-expr* is the NULL value are not added to the list.

**LIST( DISTINCT column-name )** Returns a string containing a comma-separated list composed of all the different values for *string-expr* in each group of rows. Rows where *string-expr* is the NULL value are not added to the list.

**MAX( expression )** Returns the maximum *expression* value found in each group of rows. Rows where *expression* is the NULL value are ignored. Returns the NULL value for a group containing no rows.

**MAX( DISTINCT column-name )** Returns the same as *MAX(expression)*, and is included for completeness.

**MIN( expression )** Returns the minimum *expression* value found in each group of rows. Rows where *expression* is the NULL value are ignored. Returns the NULL value for a group containing no rows.

**MIN( DISTINCT column-name )** Returns the same as *MIN( expression )*, and is included for completeness.

**SUM( expression )** Returns the total of *expression* for each group of rows. Rows where the *expression* is the NULL value are not included. Returns NULL for a group containing no rows.

**SUM( DISTINCT column-name )** Computes the sum of the unique values for *numeric-expr* for each group of rows. This is of limited usefulness, but is included for completeness.

# Numeric functions

**Purpose** Numeric functions perform mathematical operations on numerical data types or return numeric information.

**Syntax** *Numeric function:*

- ABS** ( *numeric-expr* )
- | **ACOS** ( *numeric-expr* )
- | **ASIN** ( *numeric-expr* )
- | **ATAN** ( *numeric-expr* )
- | **ATAN2** ( *numeric-expr, numeric-expr* )
- | **CEILING** ( *numeric-expr* )
- | **COS** ( *numeric-expr* )
- | **COT** ( *numeric-expr* )
- | **DEGREES** ( *numeric-expr* )
- | **EXP** ( *numeric-expr* )
- | **FLOOR** ( *numeric-expr* )
- | **LOG** ( *numeric-expr* )
- | **LOG10** ( *numeric-expr* )
- | **MOD** ( *dividend, divisor* )
- | **PI** ( \* )
- | **POWER** ( *numeric-expr, numeric-expr* )
- | **RADIANS** ( *numeric-expr* )
- | **RAND** ( [*integer-expr*] )
- | **REMAINDER** ( *numeric-expr, numeric-expr* )
- | **ROUND** ( *numeric-expr, integer-expr* )
- | **SIGN** ( *numeric-expr* )
- | **SIN** ( *numeric-expr* )
- | **SQRT** ( *numeric-expr* )
- | **TAN** ( *numeric-expr* )
- | **"TRUNCATE"** ( *numeric-expr, integer-expr* )

**See also** Aggregate functions  
String functions  
Date and time functions  
Data type conversion functions  
System functions  
Miscellaneous functions

**Description**

**ABS( numeric-expr )** Returns the absolute value of *numeric-expr*.

**ACOS( numeric-expr )** Returns the arc-cosine of *numeric-expr* in radians.

**ASIN( numeric-expr )** Returns the arc-sine of *numeric-expr* in radians.

- ATAN( numeric-expr )** Returns the arc-tangent of *numeric-expr* in radians.
- ATAN2( numeric-expr1, numeric-expr2 )** Returns the arc-tangent of *numeric-expr1/numeric-expr2* in radians.
- CEILING( numeric-expr )** Returns the ceiling (smallest integer not less than) of *numeric-expr*.
- COS( numeric-expr )** Returns the cosine of *numeric-expr*, expressed in radians.
- COT( numeric-expr )** Returns the cotangent of *numeric-expr*, expressed in radians.
- DEGREES( numeric-expr )** Converts *numeric-expr*, from radians to degrees.
- EXP( numeric-expr )** Returns the exponential function of *numeric-expr*.
- FLOOR( numeric-expr )** Returns the floor (largest integer not greater than) of *numeric-expr*.
- LOG( numeric-expr )** Returns the logarithm of *numeric-expr*.
- LOG10( numeric-expr )** Returns the logarithm base 10 of *numeric-expr*.
- MOD( dividend, divisor )** Returns the remainder when *dividend* is divided by *divisor*. Division involving a negative *dividend* will give a negative or zero result. The sign of the *divisor* has no effect.
- PI( \* )** Returns the numeric value PI.
- POWER ( numeric-expr1, numeric-expr2 )** Raises *numeric-expr1* to the power *numeric-expr2*.
- RADIANS ( numeric-expr )** Converts *numeric-expr*, from degrees to radians.
- RAND ( [ integer-expr ] )** Returns a random number in the interval 0 to 1, with *integer-expr* as an optional seed.
- REMAINDER( dividend, divisor )** Same as the MOD function.
- ROUND ( numeric-expr, integer-expr )** Rounds *numeric-expr* to *integer-expr* places after the decimal point. A positive integer determines the number of significant digits to the right of the decimal point; a negative integer, the number of significant digits to the left of the decimal point.
- SIGN( numeric-expr )** Returns the sign of *numeric-expr*.

**SIN( numeric-expr )** Returns the sine of *numeric-expr*, expressed in radians.

**SQRT( numeric-expr )** Returns the square root of *numeric-expr*.

**TAN( numeric-expr )** Returns the tangent of *numeric-expr*, expressed in radians.

**"TRUNCATE" ( numeric-expr, integer-expr )** Truncates *numeric-expr* at *integer-expr* places after the decimal point. A positive integer determines the number of significant digits to the right of the decimal point; a negative integer, the number of significant digits to the left of the decimal point.

The double quotes are required because of a keyword conflict with the TRUNCATE TABLE statement.



# String functions

## Purpose

String functions perform conversion, extraction or manipulation operations on strings or return information about strings.

When working in a multi-byte character set, check carefully whether the function being used returns information concerning characters or bytes.

## Syntax

String function:

```

ASCII ( string-expr )
| BYTE_LENGTH ( string-expr )
| BYTE_SUBSTR ( string-expr, integer-expr [, integer-expr ] )
| CHAR ( string-expr )
| DIFFERENCE ( string-expr, string-expr )
| INSERTSTR ( numeric-expr, string-expr, string-expr )
| LCASE ( string-expr )
| LEFT ( string-expr, numeric-expr )
| LENGTH ( string-expr )
| LOCATE ( string-expr, string-expr [, numeric-expr ] )
| LTRIM ( string-expr )
| PATINDEX ( '%pattern%', string_expr )
| REPEAT ( string-expr, numeric-expr )
| RIGHT ( string-expr, numeric-expr )
| RTRIM ( string-expr )
| SIMILAR ( string-expr, string-expr )
| SOUNDEX ( string-expr )
| SPACE ( integer-expr )
| STRING ( string-expr [, ...] )
| SUBSTR ( string-expr, integer-expr [, integer-expr] )
| TRIM ( string-expr )
| UCASE ( string-expr )

```

## See also

Aggregate functions  
 Numeric functions  
 Date and time functions  
 Data type conversion functions  
 System functions  
 Miscellaneous functions

## Description

**ASCII( *string-expr* )** Returns the integer ASCII value of the first byte in *string-expr*, or 0 for the empty string.

**BYTE\_LENGTH( *string-expr* )** Returns the number of bytes in the string *string-expr*.

**BYTE\_SUBSTR( string-expr, start [, length] )** Returns the substring of *string-expr* starting at the given *start* position (origin 1), in bytes. If the *length* is specified, the substring is restricted to that number of bytes. Both *start* and *length* can be negative. A negative starting position specifies a number of bytes from the end of the string instead of the beginning. A positive *length* specifies that the substring ends *length* bytes to the right of the starting position, while a negative *length* specifies that the substring ends *length* bytes to the left of the starting position. Using appropriate combinations of negative and positive numbers, you can get a substring from either the beginning or end of the string.

**CHAR( numeric-expr )** Returns the character with the ASCII value *numeric-expr*. The character in the current character set corresponding to the supplied numeric expression modulo 256 is returned. If you are using multi-byte character sets, CHAR may not return a valid character.

**DIFFERENCE( string-expr1, string-expr2 )** Returns the difference in the soundex values of *string-expr1* and *string-expr2*.

**INSERTSTR( numeric-expr, string-expr1, string-expr2)** Inserts *string-expr2* in *string-expr1* at character position *numeric-expr*.

**LCASE( string-expr )** Converts all characters in *string-expr* to lower case.

**LEFT( string-expr, numeric-expr )** Returns the leftmost *numeric-expr* characters of *string-expr*.

**LENGTH( string-expr )** Returns the number of characters in the string *string-expr*. If *string-expr* is of binary data type, the LENGTH function behaves as BYTE\_LENGTH.

**LOCATE( string-expr1, string-expr2 [, numeric-expr ] )** Returns the character offset (base 1) into the string *string-expr1* of the first occurrence of the string *string-expr2*. If *numeric-expr* is specified, the search will start at that offset into the string.

The first string can be a long string (longer than 255 bytes), but the second is limited to 255 bytes. If a long string is given as the second argument, the function returns a NULL value. If the string is not found, 0 is returned. Searching for a zero-length string will return 1. If any of the arguments are NULL, the result is NULL.

**LTRIM( string-expr )** Returns *string-expr* with leading blanks removed.

**PATINDEX( '%pattern%', string-expr )** Returns an integer representing the starting position in characters of the first occurrence of *pattern* in the specified string expression, or a zero if *pattern* is not found. If the leading percent wild card is omitted, PATINDEX returns one (1) if *pattern* occurs at the beginning of the string, and zero if not. If the trailing percent wild card is omitted, PATINDEX returns one (1) if *pattern* occurs at the end of the string, and zero if not. If *pattern* starts with a percent wild card, then the two leading percent wild cards are treated as one.

**REPEAT( string-expr, integer-expr )** Returns a string comprised of *integer-expr* instances of *string-expr*, concatenated together.

**RIGHT( string-expr, numeric-expr )** Returns the rightmost *numeric-expr* characters of *string-expr*.

**RTRIM( string-expr )** Returns *string-expr* with trailing blanks removed.

**SIMILAR( string-expr1, string-expr2 )** Returns an integer between 0 and 100 representing the similarity between the two strings. The result can be interpreted as the percentage of characters matched between the two strings (100 percent match if the two strings are identical).

This function can be very useful for correcting a list of names (such as customers). Some customers may have been added to the list more than once with slightly different names. Join the table to itself and produce a report of all similarities greater than 90 percent but less than 100 percent.

**SOUNDEX( string-expr )** Returns a number representing the sound of the *string-expr*. Although it is not perfect, soundex will normally return the same number for words which sound similar and start with the same letter. For example:

```
soundex( 'Smith' ) = soundex( 'Smythe' )
```

The soundex function value for a string is based on the first letter and the next three consonants other than H, Y, and W. Doubled letters are counted as one letter. For example,

```
soundex( 'apples' )
```

is based on the letters A, P, L and S. Multi-byte characters are ignored by the SOUNDEX function.

**STRING( string1, [ string2, ..., string99 ] )** Concatenates the strings into one large string. NULL values are treated as empty strings ("). Any numeric or date parameters are automatically converted to strings before concatenation. The STRING function can also be used to force any single expression to be a string by supplying that expression as the only parameter.

**SUBSTR( string-expr, start [, length] )** Returns the substring of *string-expr* starting at the given *start* position (origin 1). If the *length* is specified, the substring is restricted to that length. Both *start* and *length* can be negative. A negative starting position specifies a number of characters from the end of the string instead of the beginning. A positive *length* specifies that the substring ends *length* characters to the right of the starting position, while a negative *length* specifies that the substring ends *length* characters to the left of the starting position. Using appropriate combinations of negative and positive numbers, you can easily get a substring from either the beginning or end of the string. If *string-expr* is of binary data type, the SUBSTR function behaves as BYTE\_SUBSTR.

**TRIM( string-expr )** Returns *string-expr* with both leading and trailing blanks removed.

**UCASE( string-expr )** Converts all characters in *string-expr* to uppercase.

# Date and time functions

## Purpose

Date and time functions perform conversion, extraction or manipulation operations on date and time data types and can return date and time information.

## Syntax

Date and time function:

```

| DATE ( expression )
| DATEFORMAT ( datetime-expr, string-expr )
| DATENAME ( datepart, date-expr )
| DATETIME ( expression )
| DAY ( date-expr )
| DAYNAME( date-expr )
| DAYS ( date-expr )
| DAYS ( date-expr, date-expr )
| DAYS ( date-expr, integer-expr )
| DOW ( date-expr )
| HOUR ( datetime-expr )
| HOURS ( datetime-expr )
| HOURS ( datetime-expr, datetime-expr )
| HOURS ( datetime-expr, integer-expr )
| MINUTE ( datetime-expr )
| MINUTES ( datetime-expr )
| MINUTES ( datetime-expr, datetime-expr )
| MINUTES ( datetime-expr, integer-expr )
| MONTH ( date-expr )
| MONTHNAME ( date-expr )
| MONTHS ( date-expr )
| MONTHS ( date-expr, date-expr )
| MONTHS ( date-expr, integer-expr )
| NOW ( * )
| QUARTER( date-expr )
| SECOND ( expression )
| SECONDS ( datetime-expr )
| SECONDS ( datetime-expr, datetime-expr )
| SECONDS ( datetime-expr, integer-expr )
| TODAY ( * )
| WEEKS ( date-expr )
| WEEKS ( date-expr, date-expr )
| WEEKS ( date-expr, integer-expr )
| YEAR ( date-expr )
| YEARS ( date-expr )
| YEARS ( date-expr, date-expr )

```

| **YEARS** ( *date-expr*, *integer-expr* )  
| **YMD** ( *integer-expr*, *integer-expr*, *integer-expr* )

**See also**

Aggregate functions  
Numeric functions  
String functions  
Data type conversion functions  
System functions  
Miscellaneous functions

**Description**

The date and time functions allow manipulation of time units. Most time units (such as MONTH) have four functions for time manipulation, although only two names are used (such as MONTH and MONTHS).

SQL Anywhere also supports several Transact-SQL date and time functions, allowing an alternative way of accessing and manipulating date and time functions. For information about the Transact-SQL date and time functions, see "Compatibility of date and time functions" in the chapter "Using Transact-SQL with SQL Anywhere".

Arguments to date functions should be converted to dates before being used, so that

```
days ( '1995-11-17', 2 )
```

is not correct, but

```
days ( date( '1995-11-17' ), 2 )
```

is correct.

**DATE( expression )** Converts the expression into a date, and removes any hours, minutes or seconds. Conversion errors may be reported.

**DATEFORMAT( date-expr, string-expr )** Returns a string representing the date *date-expr* in the format specified by *string-expr*. Any allowable date format can be used for *string-expr*.

For example,

```
DATEFORMAT('1989-01-01', 'Mmm Dd, yyyy')
```

is

```
'Jan 1, 1989'
```

**Year 2000 compliance**

It is possible in using the DATEFORMAT function to produce a string with the year value represented by only two digits. This can cause problems with year 2000 compliance even though no error has occurred.

☞ For more information on year 2000 compliance, please see "Date to string conversions" in the chapter "SQL Anywhere Data Types".

☞ For more information, see the DATE\_FORMAT option in "SET OPTION statement" in the chapter "Watcom-SQL Statements".

**DATENAME ( datepart, date )** Returns the name of the specified part (such as the month "June") of a DATETIME value, as a character string. If the result is numeric, such as 23 for the day, it is still returned as a character string. For example, the following statement displays the value May.

```
SELECT datename( month , '1987/05/02' )
```

**DATETIME( expression )** Converts the expression into a timestamp. Conversion errors may be reported.

**DAY( date-expr )** Returns a number from 1 to 31 corresponding to the day of the given date.

**DAYNAME( date-expr )** Returns the name of the day from the supplied date expression. For example, with the date\_order option set to the default value of *ymd*:

```
SELECT DAYNAME ( '1987/05/02' )
```

returns the value Saturday.

**DAYS( datetime-expr )** Return the number of days since an arbitrary starting date.

**DAYS( date-expr, date-expr )** Returns the number of days from the first date to the second date. The number may be negative. Hours, minutes and seconds are ignored.

**DAYS( date-expr, integer-expr )** Add *integer-expr* days to the given date. If the *integer-expr* is negative, the appropriate number of days are subtracted from the date. Hours, minutes and seconds are ignored.

**DOW( date-expr )** Returns a number from 1 to 7 representing the day of the week of the given date, with Sunday=1, Monday=2, and so on.

**HOUR( datetime-expr )** Returns a number from 0 to 23 corresponding to the hour component of the given date.

**HOURS( datetime-expr )** Return the number of hours since an arbitrary starting date and time.

**HOURS( datetime-expr, datetime-expr )** Returns the number of whole hours from the first date/time to the second date/time. The number may be negative.

**HOURS( datetime-expr, integer-expr )** Add *integer-expr* hours to the given date/time. If the *integer-expr* is negative, the appropriate number of hours are subtracted from the date/time.

**MINUTE( datetime-expr )** Returns a number from 0 to 59 corresponding to the minute component of the given date/time.

**MINUTES( datetime-expr )** Return the number of minutes since an arbitrary starting date and time.

**MINUTES( datetime-expr, datetime-expr )** Returns the number of whole minutes from the first date/time to the second date/time. The number may be negative.

**MINUTES( datetime-expr, integer-expr )** Add *integer-expr* minutes to the given date/time. If the *integer-expr* is negative, the appropriate number of minutes are subtracted from the date/time.

**MONTH( date-expr )** Returns a number from 1 to 12 corresponding to the month of the given date.

**MONTHNAME( date-expr )** Returns the name of the month from the supplied date expression. For example, with the `date_order` option set to the default value of `ymd`:

```
SELECT MONTHNAME ( '1987/05/02' )
```

returns the value *May*.

**MONTHS( datetime-expr )** Return the number of months since an arbitrary starting date. This number is often useful for determining if two date/time expressions are on the same month in the same year.

```
MONTHS( invoice_sent ) = MONTHS( payment_received )
```

Note that comparing the `MONTH` function would be wrong if a payment were made 12 months after the invoice was sent.



**MONTHS( date-expr, date-expr )** Returns the number of whole months from the first date to the second date. The number may be negative. Hours, minutes and seconds are ignored.

**MONTHS( date-expr, integer-expr )** Add *integer-expr* months to the given date. If the new date is past the end of the month (such as MONTHS('1992-01-31', 1) ) the result is set to the last day of the month. If the *integer-expr* is negative, the appropriate number of months are subtracted from the date. Hours, minutes and seconds are ignored.

**NOW( \* )** Returns the current date and time. This is the historical syntax for CURRENT\_TIMESTAMP.

**QUARTER( date-expr )** Returns the quarter from the supplied date expression. For example, with the date\_order option set to the default value of *ymd*:

```
SELECT QUARTER ( '1987/05/02' )
```

returns the value 2.

**SECOND( datetime-expr )** Returns a number from 0 to 59 corresponding to the second component of the given date.

**SECONDS( datetime-expr )** Return the number of seconds since an arbitrary starting date and time.

**SECONDS( datetime-expr, datetime-expr )** Returns the number of whole seconds from the first date/time to the second date/time. The number may be negative.

**SECONDS( datetime-expr, integer-expr )** Add *integer-expr* seconds to the given date/time. If the *integer-expr* is negative, the appropriate number of seconds are subtracted from the date/time.

**TODAY( \* )** Returns today's date. This is the historical syntax for CURRENT\_DATE.

**WEEKS( datetime-expr )** Return the number of weeks since an arbitrary starting date. (Weeks are defined as going from Sunday to Saturday, as they do in a North American calendar.) This number is often useful for determining if two dates are in the same week.

```
WEEKS( invoice_sent ) = WEEKS( payment_received )
```

**WEEKS( date-expr, date-expr )** Returns the number of whole weeks from the first date to the second date. The number may be negative. Hours, minutes and seconds are ignored.

**WEEKS( date-expr, integer-expr )** Add *integer-expr* weeks to the given date. If the *integer-expr* is negative, the appropriate number of weeks are subtracted from the date. Hours, minutes and seconds are ignored.

**YEAR( date-expr )** Returns a 4 digit number corresponding to the year of the given date.

**YEARS( date-expr )** Same as the YEAR function.

**YEARS( date-expr, date-expr )** Returns the number of whole years from the first date to the second date. The number may be negative. Hours, minutes and seconds are ignored. For example, age can be calculated by

```
YEARS( birthdate, CURRENT DATE )
```

**YEARS( date-expr, integer-expr )** Add *integer-expr* years to the given date. If the new date is past the end of the month (such as `YEARS( '1992-02-29', 1 )`) the result is set to the last day of the month. If the *integer-expr* is negative, the appropriate number of years are subtracted from the date. Hours, minutes, and seconds are ignored.

**YMD( year-num, month-num, day-num )** Returns a date value corresponding to the given year, month, and day of the month. If the month is outside the range 1-12, the year is adjusted accordingly. Similarly, the day is allowed to be any integer: the date is adjusted accordingly. For example,

```
YMD( 1992, 15, 1 ) = 'Mar 1 1993'
```

```
YMD( 1992, 15, 1-1 ) = 'Feb 28 1993'
```

```
YMD( 1992, 3, 1-1 ) = 'Feb 29 1992'
```

## Data type conversion functions

<b>Purpose</b>	Data type conversion functions convert values.
<b>Syntax</b>	Data type conversion function: <b>CAST</b> ( <i>expression AS datatype</i> ) <b>CONVERT</b> ( <i>datatype, expression [ , format-style ]</i> )
<b>See also</b>	Aggregate functions Numeric functions String functions Date and time functions System functions Miscellaneous functions
<b>Description</b>	<p>The DATE, DATETIME, and DATEFORMAT functions which convert expressions to dates, timestamps, or strings based on a date format are listed in "Date and time functions" on page 945. The STRING function, which converts expressions to a string, is discussed in "String functions" on page 941.</p> <p>SQL Anywhere carries out many type conversions automatically. For example, if a string is supplied where a numerical expression is required, the string is automatically converted to a number. For more information on automatic data type conversions carried out by SQL Anywhere, see "Data type conversions" in the chapter "SQL Anywhere Data Types".</p> <p><b>CAST( expression AS data-type )</b> Returns the value of <i>expression</i>, converted to the supplied <i>data-type</i>. If the length is omitted for character string types, SQL Anywhere chooses an appropriate length. If neither precision nor scale is specified for a DECIMAL conversion, SQL Anywhere selects appropriate values. For example:</p> <pre>CAST( '1992-10-31' AS DATE ) --ensure string is used --as a DATE  CAST( 1 + 2 AS CHAR ) --SQL Anywhere chooses --length  CAST( Surname AS CHAR(10) ) --useful for shortening --strings</pre> <p><b>CONVERT( data-type, expression )</b> Returns the expression converted to data-type. The following is an example of this use of CONVERT:</p> <pre>SELECT CONVERT (VARCHAR(12), order_date ) FROM sales_order ;</pre>

**CONVERT( data-type, expression, format-style )** For converting strings to date or time data types and vice versa, the *format-style* is a style code number describing the date format string to be used. The values of the *format-style* argument have the following meanings:

(yy)	(yyyy)	Output
-	-0 or 100	mon dd yyyy hh:miAM (or PM)
1	101	mm/dd/yy
2	102	yy.mm.dd
3	103	dd/mm/yy
4	104	dd.mm.yy
5	105	dd-mm-yy
6	106	dd mon yy
7	107	mon dd, yy
8	108	hh:mm:ss
-	9 or 109	mmm dd yyyy hh:mi:ss:mmmAM (or PM)
10	110	mm-dd-yy
11	111	yy/mm/dd
12	112	yymmdd

If no *format-style* argument is provided, Style Code 0 is used.

# System functions

## Purpose

System functions return system information.

## Syntax

System function:

```

connection_property ( { integer-expr | string-expr }
                    ... [ , integer-expr ] )
| datalength ( expression )
| db_id ( [ string-expr ] )
| db_name ( [ integer-expr ] )
| db_property ( { integer-expr | string-expr }
                ... [, { integer-expr | string-expr } ] )
| next_connection ( { NULL | string-expr } )
| next_database ( { NULL | string-expr } )
| property ( { integer-expr | string-expr } )
| property_name ( integer-expr )
| property_number ( string-expr )
| property_description ( { integer-expr | string-expr } )

```

## See also

Aggregate functions  
 Numeric functions  
 String functions  
 Date and time functions  
 Data type conversion functions  
 Miscellaneous functions

## Description

Databases currently running on a server or engine are identified by a database name and a database id number. The `db_id` and `db_name` functions provide information on these values.

A set of system functions provides information about properties of a currently running database, or of a connection, on the database engine. These system functions take the database name or ID, or the connection name, as an optional argument to identify the database or connection for which the property is requested.

The available properties and their uses are described following the listing of the functions.

**connection\_property** ( { **property-id** | **property-name** } [, **connection-id** ] ) Returns the value of the given property as a string. The current connection is used if the second argument is omitted.

**datalength ( expression )** Returns the length of the expression in bytes. The expression is usually a column name. If the expression is a string constant, it must be enclosed in quotes. The following query displays the longest string in the *company\_name* column of the *customer* table:

```
SELECT MAX( DATALENGTH( company_name ) )
FROM customer
```

**db\_id ( [database-name] )** Returns the database ID number. The supplied *database\_name* must be a string expression; if it is a constant expression, it must be enclosed in quotes. If no *database\_name* is supplied, the ID number of the current database is returned.

**db\_name ( [database-id] )** Returns the database name. The supplied *database\_id* must be a numeric expression. If no *database\_id* is supplied, the name of the current database is returned.

**db\_property ( { property-id | property-name } [, { database-id | database-name } ] )** Returns the value of the given property as a string. The current database is used if the second argument is omitted.

**next\_connection ( { NULL | connection-id } )** Returns the next connection number, or the first connection if parm is NULL.

**next\_database ( { NULL | database-id } )** Returns the next database number, or the first connection if parm is NULL.

**property ( { property-number | property-name } )** Returns the value of the specified property as a string.

**property\_name ( property-number )** Returns the name of the property with the supplied property-number.

**property\_number ( property-name )** Returns the number of the property with the supplied property-name.

**property\_description ( { property-number | property-name } )**  
Returns a description of the property with the supplied property-name or property-number. The statistics and properties available are those in the following list. The list includes the name of each property, and a brief description. While each property does have a number as well as a name, the number is subject to change between releases of SQL Anywhere, and should not be used as a reliable identifier for a given property.

## Connection properties

The following table lists properties available for each connection.

### Examples

#### ❖ To retrieve the value of a connection property:

- ◆ Use the `connection_property` system function: The following statement returns the number of pages that have been read from file by the current connection.

```
select connection_property ( 'DiskRead' )
```

#### ❖ To retrieve the values of all connection properties:

- ◆ Use the `sa_conn_properties` system procedure:

```
call sa_eng_properties
```

A separate row is displayed for each connection, for each property.

### Descriptions

Property	Description
Async2Read	The number of rereads. A reread occurs when a read request for a page is received by the database IO subsystem while an asynchronous read IO operation has been posted to the operating system but has not completed.
AsyncRead	The number of pages that have been read asynchronously from disk.
AsyncWrite	The number of pages that have been written asynchronously to disk.
BlockedOn	If the current connection is not blocked this is zero. If it is blocked, the connection number on which the connection is blocked due to a locking conflict.
CacheRead	The number of database pages that have been looked up in the cache.
CacheReadIndInt	The number of index internal-node pages that have been read from the cache.
CacheReadIndLeaf	The number of index leaf pages that have been read from the cache.
CacheReadTable	The number of table pages that have been read from the cache.
CacheWrite	The number of pages in the cache that have been modified.
Commit	The number of Commit requests that have been handled.

Property	Description
CommLink	The communication link for the connection. This is one of the network protocols supported by SQL Anywhere, or is "local" for a connection without a SQL Anywhere Client.
CurrTaskSwitch	The number of current request context switches.
Cursors	The number of declared cursors that are currently being maintained by the engine.
CursorOpen	Open cursors is the number of open cursors that are currently being maintained by the engine.
DBNumber	The id number of the database.
DiskRead	The number of pages that have been read from file.
DiskReadIndInt	The number of index internal-node pages that have been read from disk.
DiskReadIndLeaf	The number of index leaf pages that have been read from disk.
DiskReadTable	The number of table pages that have been read from disk.
DiskSyncRead	The number of pages that have been read synchronously from disk..
DiskSyncWrite	The number of pages that have been written synchronously to disk. It is the sum of all the other "Disk SyncWrites" counters.
DiskWaitRead	The number of times the engine has waited synchronously for the completion of a read IO operation which was originally issued as an asynchronous read. Waitreads often occur due to cache misses on systems that support asynchronous IO.
DiskWaitWrite	The number of times engine has waited synchronously for the completion of a write IO operation which was originally issued as an asynchronous write.
DiskWrite	The number of modified pages that have been written to disk.
FullCompare	The number of comparisons beyond the hash value in an index that have been performed.
HintUsed	The number of page-read operations that have been satisfied immediately from cache thanks to a earlier read hint.
IndAdd	The number of entries that have been added to indexes.



Property	Description
IndLookup	The number of entries that have been looked up in indexes.
LastIdle	The number of ticks between requests.
LastReqTime	The time at which the last request for the specified connection started.
LockTablePages	The number of pages used for the lock table.
LogFreeCommit	The number of Redo Free Commits. A "Redo Free Commit" occurs when a commit of the transaction log is requested but the log has already been written (so the commit was done for "free").
LogRewrite	The number of pages that were previously written to the transaction log (but were not full) that have been written to the transaction log again (but with more data added).
LogWrite	The number of pages that have been written to the transaction log.
Name	The database name.
NodeAddress	The node for the client in a client/server connection.
Number	The ID number of the connection.
Port	An application-specific number for each client machine, identifying the connection port.
PrepStmt	The number of prepared statements that are currently being maintained by the engine.
ProcessTime	The time since the start of the connection.
ReadHint	The number of read hints. A read hint is an asynchronous read operation for a page that the database engine is likely to need soon.
ReqType	A string for the type of the last request.
Rlbk	The number of Rollback requests that have been handled.
RollbackLogPages	The number of pages in the rollback log
SyncWriteChkpt	The number of pages that have been written synchronously to disk for a checkpoint.
SyncWriteExtend	The number of pages that have been written synchronously to disk while extending a database file.
SyncWriteFreeCurr	The number of pages that have been written synchronously to disk to free a page that cannot remain in the in-memory free list.

Property	Description
SyncWriteFreePush	The number of pages that have been written synchronously to disk to free a page that can remain in the in-memory free list.
SyncWriteLog	The number of pages that have been written synchronously to the transaction log.
SyncWriteRlbc	The number of pages that have been written synchronously to the rollback log.
SyncWriteUnkn	The number of pages that have been written synchronously to disk for a reason not covered by other "Disk SyncWrites" counters.
TaskSwitch	The number of times the current engine thread has been changed.
TaskSwitchCheck	The number of times the current engine thread has volunteered to give up the CPU to another engine thread.
UncommitOp	The number of uncommitted operations for the connection.
Userid	The user ID for the connection.
VoluntaryBlock	The number of engine threads that have voluntarily blocked on pending disk IO.
WaitReadCmp	The number of read requests associated with a full comparison (a comparison beyond the hash value in an index) that must be satisfied by a synchronous read operation.
WaitReadOpt	The number of read requests posted by the optimizer that must be satisfied by a synchronous read operation.
WaitReadSys	The number of read requests posted from the system connection that must be satisfied by a synchronous read operation. The system connection is a special connection used as the context before a connection is made and for operations performed outside of any client connection.
WaitReadTemp	The number of read requests for a temporary table that must be satisfied by a synchronous read operation.
WaitReadUnkn	The number of read requests from other sources that must be satisfied by a synchronous read operation.

## Properties available for the engine

The following table lists properties that apply across the engine as a whole.

### Examples

#### ❖ To retrieve the value of an engine property:

- ◆ Use the property system function: The following statement returns the number of cache pages being used to hold the main heap.

```
select property ( 'MainHeapPages' )
```

#### ❖ To retrieve the values of all engine properties:

- ◆ Use the sa\_eng\_properties system procedure:

```
call sa_eng_properties
```

### Descriptions

Property	Description
ActiveReq	The number of engine threads that are currently handling a request.
Async2Read	The number of rereads. A reread occurs when a read request for a page is received by the database IO subsystem while an asynchronous read IO operation has been posted to the operating system but has not completed.
AsyncRead	The number of pages that have been read asynchronously from disk.
AsyncWrite	The number of pages that have been written asynchronously to disk.
CacheHits	The number of database page lookups satisfied by finding the page in the cache.
CacheRead	The number of database pages that have been looked up in the cache.
CacheReadIndInt	The number of index internal-node pages that have been read from the cache.
CacheReadIndLeaf	The number of index leaf pages that have been read from the cache.
CacheReadTable	The number of table pages that have been read from the cache.
CacheWrite	The number of pages in the cache that have been modified.
Chkpt	The number of checkpoints that have been performed.

Property	Description
ChkptFlush	The number of ranges of adjacent pages written out during a checkpoint.
ChkptPage	The number of transaction log checkpoints.
CommitFile	The number of times the engine has forced a flush of the disk cache. On NT and NetWare platforms, the disk cache does not need to be when unbuffered (direct) IO is used.
CompanyName	The name of the company owning this software (Sybase, Inc.).
ContReq	The number of "CONTINUE" requests issued to the engine.
CurrIO	The current number of file IOs issued by the engine which have not yet completed.
CurrRead	The current number of file reads issued by the engine which have not yet completed.
CurrWrite	The current number of file writes issued by the engine which have not yet completed.
DiskRead	The number of pages that have been read from file.
DiskReadIndInt	The number of index internal-node pages that have been read from disk.
DiskReadIndLeaf	The number of index leaf pages that have been read from disk.
DiskReadTable	The number of table pages that have been read from disk.
DiskSyncRead	The number of pages that have been read synchronously from disk.
DiskSyncWrite	The number of pages that have been written synchronously to disk. It is the sum of all the other "Disk SyncWrites" counters.
DiskWaitRead	The number of times the engine has waited synchronously for the completion of a read IO operation which was originally issued as an asynchronous read. Waitreads often occur due to cache misses on systems that support asynchronous IO.
DiskWaitWrite	The number of times engine has waited synchronously for the completion of a write IO operation which was originally issued as an asynchronous write.
DiskWrite	The number of modified pages that have been written

Property	Description
ExtendDBWrite	to disk. The number of pages by which the database file has been extended.
ExtendTempWrite	The number of pages by which temporary files have been extended.
FreeWriteCurr	The number of pages freed of those that cannot remain in the in-memory free list.
FreeWritePush	The number of pages freed of those that can remain in the in-memory free list.
FullCompare	The number of comparisons beyond the hash value in an index that have been performed.
HintUsed	The number of page-read operations that have been satisfied immediately from cache thanks to a earlier read hint.
IdleCheck	The number of times the engine's idle thread has become active to do idle writes, idle checkpoints, and so on.
IdleChkpt	The number of checkpoints completed by the engine's idle thread. An idle checkpoint occurs whenever the idle thread writes out the last dirty page in the cache.
IdleChkTime	The number of 100'ths of a second spent checkpointing during idle I/O.
IdleWrite	The number of disk writes that have been issued by the engine's idle thread.
IndAdd	The number of entries that have been added to indexes.
IndLookup	The number of entries that have been looked up in indexes.
LegalCopyright	The Copyright string for the software.
LegalTrademarks	Trademark information for the software.
LockTablePages	The number of pages used to store lock information
LogFreeCommit	The number of Redo Free Commits. A "Redo Free Commit" occurs when a commit of the transaction log is requested but the log has already been written (so the commit was done for "free").
LogRewrite	The number of pages that were previously written to the transaction log (but were not full) that have been written to the transaction log again (but with more data

Property	Description
	added).
LogWrite	The number of pages that have been written to the transaction log.
MainHeapPages	The number of pages used for global engine data structures.
MapPages	The number of map pages used for accessing the lock table, frequency table, and table layout.
MaxIO	The maximum value that "Current IO" has reached.
MaxRead	The maximum value that "Current Reads" has reached.
MaxWrite	The maximum value that "Current Writes" has reached.
Name	The name of the engine or server.
PageRelocations	The number of relocatable heap pages read from the temporary file.
PendingReq	The number of new requests detected by the engine.
Platform	The operating system on which the software is running.
ProcedurePages	The number of relocatable heap pages used for procedures.
ProductName	The name of the software.
ProductVersion	The version of the software being run.
ReadHint	The number of read hints. A read hint is an asynchronous read operation for a page that the database engine is likely to need soon.
RelocatableHeapPages	The number of pages used for relocatable heaps (cursors, statements, procedures, triggers, views, etc.).
Req	The number of times the engine has been entered to allow it to handle a new request or continue processing an existing request.
RollbackLogPages	The number of pages in the rollback log.
SyncWriteChkpt	The number of pages that have been written synchronously to disk for a checkpoint.
SyncWriteExtend	The number of pages that have been written synchronously to disk while extending a database file.
SyncWriteFreeCurr	The number of pages that have been written synchronously to disk to free a page that cannot remain in the in-memory free list.

Property	Description
SyncWriteFreePush	The number of pages that have been written synchronously to disk to free a page that can remain in the in-memory free list.
SyncWriteLog	The number of pages that have been written synchronously to the transaction log.
SyncWriteRlbk	The number of pages that have been written synchronously to the rollback log.
SyncWriteUnkn	The number of pages that have been written synchronously to disk for a reason not covered by other "Disk SyncWrites" counters.
TriggerPages	The number of relocatable heap pages used for triggers.
UnschReq	The number of requests that are currently queued up waiting for an available engine thread.
ViewPages	The number of relocatable heap pages used for views.
VoluntaryBlock	The number of engine threads that have voluntarily blocked on pending disk IO.
WaitReadCmp	The number of read requests associated with a full comparison (a comparison beyond the hash value in an index) that must be satisfied by a synchronous read operation.
WaitReadOpt	The number of read requests posted by the optimizer that must be satisfied by a synchronous read operation.
WaitReadSys	The number of read requests posted from the system connection that must be satisfied by a synchronous read operation. The system connection is a special connection used as the context before a connection is made and for operations performed outside of any client connection.
WaitReadTemp	The number of read requests for a temporary table that must be satisfied by a synchronous read operation.
WaitReadUnkn	The number of read requests from other sources that must be satisfied by a synchronous read operation.

## Properties available for each database

The following table lists properties available for each database on the server.

**Examples**

❖ **To retrieve the value of a database property:**

- ◆ Use the `db_property` system function: The following statement returns the page size of the current database.

```
select db_property ( 'PageSize')
```

❖ **To retrieve the values of all database properties:**

- ◆ Use the `sa_db_properties` system procedure:

```
call sa_db_properties
```

**Descriptions**

<b>Property</b>	<b>Description</b>
Alias	The database name.
ConnCount	The number of connections to the database.
File	The file name of the database root file, including path.
FileVersion	The version of the database file.
LogName	The file name of the transaction log, including path.
Name	The database name, or alias.
PageSize	The page size of the database, in bytes.



# Miscellaneous functions

<b>Purpose</b>	Miscellaneous functions perform operations on arithmetic, string or date/time expressions, including the return values of other functions.
<b>Syntax</b>	<p>Miscellaneous function:</p> <ul style="list-style-type: none"> <li>  <b>ARGN</b> ( <i>integer-expr</i>, <i>expression</i> [ , ... ] )</li> <li>  <b>COALESCE</b> ( <i>expression</i>, <i>expression</i> [ , ... ] )</li> <li>  <b>ESTIMATE</b> ( <i>column-name</i> [ , <i>number</i> [ , <i>relation-string</i> ] ] )</li> <li>  <b>ESTIMATE_SOURCE</b> ( <i>column-name</i> [ , <i>number</i> [ , <i>relation-string</i> ] ] )</li> <li>  <b>IFNULL</b> ( <i>expression</i>, <i>expression</i> [ , <i>expression</i> ] )</li> <li>  <b>INDEX_ESTIMATE</b>( <i>column-name</i>, <i>number</i> [ , <i>relation-string</i> ] )</li> <li>  <b>EXPERIENCE_ESTIMATE</b>( <i>column-name</i>, <i>number</i> [ , <i>relation-string</i> ] ] )</li> <li>  <b>ISNULL</b> ( <i>expression</i>, <i>expression</i> [ , ... ] )</li> <li>  <b>NUMBER</b> ( * )</li> <li>  <b>PLAN</b> ( <i>string-expr</i> )</li> <li>  <b>TRACEBACK</b> ( * )</li> </ul>
<b>See also</b>	<p>Aggregate functions          Numeric functions          String functions          Date and time functions          Data type conversion functions          System functions</p>
<b>Description</b>	<p><b>ARGN</b>( <i>integer-expr</i>, <b>expression</b> [ , ... ] ) Using the value of <i>integer-expr</i> as <i>n</i>, return the <i>n</i>'th argument (starting at 1) from the remaining list of arguments.</p> <p><b>COALESCE</b>( <b>expression</b>, <b>expression</b> [ ... , <b>expression</b>] ) Returns the value of the first expression that is not NULL.</p> <p><b>ESTIMATE</b>( <b>column-name</b> [ , <b>number</b> [ , <b>relation-string</b>] ] ) The <i>relation-string</i> must be a comparison operator enclosed in single quotes; the default is =. If <i>number</i> is specified, the function returns as a REAL the percentage estimate the query optimizer uses for the following condition:</p> <pre>column-name relation number</pre> <p>If <i>number</i> is not specified, the function returns the estimate used by the query-optimizer for the following condition:</p> <pre>column-name relation expression</pre> <p>The function returns NULL if the relation-string is not valid. For example, the following query returns the percentage estimate for employee ID numbers being greater than 200:</p>

```
SELECT DISTINCT ESTIMATE( emp_id, 200, '>' )
FROM employee
```

**ESTIMATE\_SOURCE( column-name [, number [, relation-string]] )**

This function is the same as the ESTIMATE function, except that it returns one of the strings Column, Value, or Index, where:

- ◆ Column means the estimate stored for the column
- ◆ Value means the estimate for a particular value, stored in the frequency table
- ◆ Index means the estimate derived from an index on the column

**EXPERIENCE\_ESTIMATE(column-name [, number [, relation-string]])**

This function is the same as the ESTIMATE function, except that it looks only in the frequency table.

**IFNULL( expression1, expression2 [, expression3] )** If the first expression is the NULL value, then the second expression is returned. Otherwise, the value of the third expression is returned if it was specified. If there was no third expression and the first expression is not NULL then the NULL value is returned.

**INDEX\_ESTIMATE( column-name [, number [, relation-string]] )** This function is the same as the ESTIMATE function, except that it looks only in an index.

**ISNULL( expression, expression [ ... , expression] )** Same as the COALESCE function.

**NUMBER( \* )** Generates numbers starting at 1 for each successive row in the results of the query. Although the NUMBER(\*) function is useful for generating primary keys when using the insert from select statement (see "INSERT statement" in the chapter "Watcom-SQL Statements"), the AUTOINCREMENT column is a preferred mechanism for generating sequential primary keys. For information on the AUTOINCREMENT default, see "CREATE TABLE statement" in the chapter "Watcom-SQL Statements".

You should not use the NUMBER( \* ) function anywhere but in a select-list. If you do use NUMBER( \* ) rather than the preferred AUTOINCREMENT, you should check your results carefully, as the behavior is not reliable in several circumstances. For example, including the function in a WHERE clause or a HAVING clause produces unpredictable results, and you should not include NUMBER( \* ) in a UNION operation.

In Embedded SQL, care should be exercised when seeking a cursor that references a query containing a `NUMBER(*)` function. In particular, this function returns negative numbers when a database cursor is positioned relative to the end of the cursor (an absolute seek with a negative offset).

**PLAN( string-expr )** Returns the optimization strategy of the `SELECT` statement *string-expr* as a string.

**TRACEBACK( \* )** Returns a string containing a traceback of the procedures and triggers that were executing when the most recent exception (error) occurred. This is useful for debugging procedures and triggers. To use the traceback function, enter the following after an error occurs while executing a procedure.

```
SELECT TRACEBACK ( * )
```



## CHAPTER 41

# Watcom-SQL Statements

### About this chapter

This chapter presents detailed descriptions of the SQL statements available in the Watcom-SQL language—one of two SQL dialects native to SQL Anywhere.

This chapter contains an alphabetical listing of all Watcom-SQL statements, including some that can only be used from Embedded SQL, and ISQL.

*↪* For information on Transact-SQL support, see the chapters "Using Transact-SQL with SQL Anywhere" and "Transact-SQL Procedure Language".

Descriptions of SQL, ISQL and Embedded SQL statements, as well as the elements that make up those statements, are included in this chapter.

## ALLOCATE DESCRIPTOR statement

<b>Function</b>	To allocate space for a SQL descriptor area (SQLDA).
<b>Syntax</b>	<b>ALLOCATE DESCRIPTOR</b> <i>descriptor-name</i> ... [ <b>WITH MAX</b> { <i>integer</i>   <i>hostvar</i> } ]
<b>Usage</b>	Embedded SQL.
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>See also</b>	DEALLOCATE DESCRIPTOR statement "The SQL descriptor area (SQLDA)" in the chapter "The Embedded SQL Interface"

**Description** Allocates space for a descriptor area (SQLDA). You must declare the following in your C code prior to using this statement:

```
struct sqlda * descriptor_name
```

The WITH MAX clause allows you to specify the number of variables within the descriptor area. The default size is one.

You must still call fill\_sqlda to allocate space for the actual data items before doing a fetch or any statement that accesses the data within a descriptor area.

**Example** The following sample program includes an example of ALLOCATE DESCRIPTOR statement usage.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

EXEC SQL INCLUDE SQLCA;

#include <sqldef.h>

EXEC SQL BEGIN DECLARE SECTION;
int          x;
short        type;
int          numcols;
char         string[100];
a_sql_statement_number stmt = 0;
EXEC SQL END DECLARE SECTION;

int main(int argc, char * argv[])
{
    struct sqlda *      sqlda1;
```

```

        if( !db_init( &sqlca ) ) {
            return 1;
        }
        db_string_connect( &sqlca,
"UID=dba;PWD=sql;DBF=d:\\sqlany50\\sample.db");

        EXEC SQL ALLOCATE DESCRIPTOR sqllda1 WITH MAX 25;

        EXEC SQL PREPARE :stmt FROM 'select * from
employee';
        EXEC SQL DECLARE curs CURSOR FOR :stmt;
        EXEC SQL OPEN curs;

        EXEC SQL DESCRIBE :stmt into sqllda1;
        EXEC SQL GET DESCRIPTOR sqllda1 :numcols=COUNT;
        // how many columns?
        if( numcols > 25 ) {
            // reallocate if necessary
            EXEC SQL DEALLOCATE DESCRIPTOR sqllda1;
            EXEC SQL ALLOCATE DESCRIPTOR sqllda1
                WITH MAX :numcols;
        }
        type = DT_STRING; // change the type to string
        EXEC SQL SET DESCRIPTOR sqllda1 VALUE 2 TYPE =
:type;
        fill_sqllda( sqllda1 ); // now we allocate space
for the variables

        EXEC SQL FETCH ABSOLUTE 1 curs USING DESCRIPTOR
sqllda1;
        EXEC SQL GET DESCRIPTOR sqllda1 VALUE 2 :string =
DATA;

        printf("name = %s", string );

        EXEC SQL DEALLOCATE DESCRIPTOR sqllda1;
        EXEC SQL CLOSE curs;
        EXEC SQL DROP STATEMENT :stmt;

        db_string_disconnect( &sqlca, "" );
        db_fini( &sqlca );

        return 0;
    }

```

## ALTER DBSPACE statement

**Function** To modify the characteristics of the main database file or an extra dbspace. To preallocate space for a database or for the transaction log.

**Syntax** **ALTER DBSPACE** { *dbspace-name* | **TRANSLOG** }  
...  
    **ADD** *number*  
    | **RENAME** *filename*

**Usage** Anywhere.

**Permissions** Must have DBA authority. Must be the only connection to the database.

**Side effects** Automatic commit.

**See also** CREATE DBSPACE statement

**Description** Each database is held in one or more files. A dbspace is an internal name associated with each database file. ALTER DBSPACE modifies the main database file (also called the root file) or an extra dbspace. The dbspace names for a database are held in the SYSFILE system table. The default dbspace name for the root file of a database is SYSTEM.

An ALTER DBSPACE with the ADD clause is used to preallocate disk space to a dbspace. It extends the size of a dbspace by the number of pages given by *number*. The page size of a database is defined when the database is created.

The ALTER DBSPACE statement with the ADD clause allows database files to be extended in large amounts before the space is required, rather than the normal 32 pages at a time when the space is needed. This can improve performance for loading large amounts of data and also serves to keep the dbspace files more contiguous within the file system.

The ALTER DBSPACE statement with the special dbspace name TRANSLOG preallocates disk space to the transaction log. Preallocation improves performance if the transaction log is expected to grow quickly. You may want to use this feature if, for example, you are handling large amounts of binary large objects (blobs), such as bitmaps.

The preallocation is carried out by altering the special dbspace name TRANSLOG, as follows:

```
ALTER DBSPACE TRANSLOG ADD number
```

This extends the size of the transaction log by the number of pages specified.



If you move a database file other than the root file to a different filename, directory, or device, use the ALTER DBSPACE statement with RENAME to ensure that SQL Anywhere can find the file when the database is started.

When a multi-file database is started, the start line or ODBC data source description tells SQL Anywhere where to find the root database file. The root database file (which has the default dbspace name SYSTEM) holds the system tables, and SQL Anywhere looks in these system tables to find the location of the other dbspaces. SQL Anywhere then opens each of the other dbspaces.

Using ALTER DBSPACE with RENAME on a root file has no effect.

**Examples**

Increase the size of the SYSTEM dbspace by 200 pages.

```
ALTER DBSPACE system
ADD 200
```

Rename the file for dbspace SYSTEM\_2 to dbspace2.

```
ALTER DBSPACE system_2
RENAME 'e:\db\dbspace2.db'
```

## ALTER PROCEDURE statement

<b>Function</b>	To replace a procedure with a modified version. You must include the entire new procedure in the ALTER PROCEDURE statement, and reassign user permissions on the procedure.
<b>Syntax</b>	<b>ALTER PROCEDURE</b> [ <i>owner</i> .] <i>procedure-name</i> ( [ <i>parameter</i> , ... ] ) ... [ <b>RESULT</b> ( <i>result-column</i> , ... ) ] ... [ <b>ON EXCEPTION RESUME</b> ] ... <i>compound-statement</i>
<b>Parameters</b>	<i>parameter</i> : <i>parameter_mode</i> <i>parameter-name</i> <i>data-type</i> ... [ <b>DEFAULT</b> <i>expression</i> ]   <b>SQLCODE</b>   <b>SQLSTATE</b>  <i>parameter_mode</i> : <b>IN</b>   <b>OUT</b>   <b>INOUT</b>  <i>result-column</i> : <i>column-name</i> <i>data-type</i>
<b>Usage</b>	Anywhere.
<b>Permissions</b>	Must be the owner of the procedure or be DBA.
<b>Side effects</b>	Automatic commit.
<b>See also</b>	CREATE PROCEDURE statement
<b>Description</b>	The ALTER PROCEDURE statement is identical in syntax to the CREATE PROCEDURE statement except for the first word. The ALTER PROCEDURE statement replaces the entire contents of the CREATE PROCEDURE statement with the contents of the ALTER PROCEDURE statement. Existing permissions on the procedure are maintained, and do not have to be reassigned. If a DROP PROCEDURE and CREATE PROCEDURE were carried out, execute permissions would have to be reassigned.

# ALTER PUBLICATION statement

<b>Function</b>	To alter the definition of a SQL Remote publication.
<b>Syntax</b>	<pre>ALTER PUBLICATION [ owner.]publication-name   ADD TABLE article-description   ...   MODIFY TABLE article-description     { DELETE   DROP } TABLE [ owner.]table-name     RENAME publication-name</pre>
<b>Parameters</b>	<p><i>article-description:</i></p> <pre>table-name [ ( column-name, ... ) ] ... [ WHERE search-condition ] ... [ SUBSCRIBE BY expression ]</pre>
<b>Usage</b>	Anywhere. This statement is applicable only to SQL Remote.
<b>Permissions</b>	Must have DBA authority, or be owner of the publication. Requires exclusive access to all tables referred to in the statement.
<b>Side effects</b>	Automatic commit.
<b>See also</b>	CREATE PUBLICATION statement DROP PUBLICATION statement
<b>Description</b>	The ALTER PUBLICATION statement alters a SQL Remote publication in the database. The contribution to a publication from one table is called an <b>article</b> . Changes can be made to a publication by adding, modifying, or deleting articles, or by renaming the publication. If an article is modified, the entire specification of the modified article must be entered.
<b>Example</b>	<p>The following statement adds the customer table to the pub_contact publication.</p> <pre>ALTER PUBLICATION pub_contact (   ADD TABLE customer )</pre>

## **ALTER REMOTE MESSAGE TYPE statement**

<b>Function</b>	To change the publisher's address for a given message system, for a message type that has been created.
<b>Syntax</b>	<b>ALTER REMOTE MESSAGE TYPE</b> <i>message-system</i> .. <b>ADDRESS</b> <i>address-string</i>
<b>Parameters</b>	<i>message-system</i> : <b>MAP   FILE   VIM   SMTP</b>
<b>Usage</b>	Anywhere. This statement is applicable only to SQL Remote.
<b>Permissions</b>	Must have DBA authority.
<b>Side effects</b>	Automatic commit.
<b>See also</b>	CREATE REMOTE MESSAGE TYPE statement
<b>Description</b>	<p>The statement changes the publisher's address for a given message type.</p> <p>The Message Agent sends outgoing messages from a database by one of the supported message links. The extraction utility uses this address when executing the GRANT CONSOLIDATE statement in the remote database.</p> <p>The address is the publisher's address under the specified message system. If it is an e-mail system, the address string must be a valid e-mail address. If it is a file-sharing system, the address string is a subdirectory of the directory specified by the SQLREMOTE environment variable, or of the current directory if that is not set. You can override this setting on the GRANT CONSOLIDATE statement at the remote database.</p> <p>For the FILE link, the ALTER REMOTE MESSAGE TYPE statement also causes the Message Agent to look for incoming messages in the address given for each message type.</p>
<b>Example</b>	<p>The following statement changes the publisher's address for the FILE message link to new_addr.</p>

```
CREATE REMOTE MESSAGE TYPE file  
ADDRESS 'new_addr'
```

# ALTER TABLE statement

## Function

To modify a table definition.

## Syntax

```
ALTER TABLE [ owner.]table-name
...
  ADD column-definition [column-constraint ...]
| ADD table-constraint
| MODIFY column-definition
| MODIFY column-name DEFAULT default-value
| MODIFY column-name [ NOT ] NULL
| MODIFY column-name CHECK NULL
| MODIFY column-name CHECK ( condition )
| { DELETE | DROP } column-name
| { DELETE | DROP } CHECK
| { DELETE | DROP } UNIQUE ( column-name, ... )
| { DELETE | DROP } PRIMARY KEY
| { DELETE | DROP } FOREIGN KEY role-name
| RENAME new-table-name
| RENAME column-name TO new-column-name
```

## Parameters

*column-definition:*

```
column-name data-type [ NOT NULL ] [ DEFAULT default-value ]
```

*column-constraint:*

```
UNIQUE
| PRIMARY KEY
| REFERENCES table-name [ ( column-name ) ] [ actions ]
| CHECK ( condition )
```

*default-value:*

```
string
| number
| AUTOINCREMENT
| CURRENT DATE
| CURRENT TIME
| CURRENT TIMESTAMP
| NULL
| USER
```

*table-constraint:*

```
UNIQUE ( column-name, ... )
| PRIMARY KEY ( column-name, ... )
| CHECK ( condition )
| foreign-key-constraint
```

*foreign-key-constraint:*

```
[ NOT NULL ] FOREIGN KEY [ role-name ] [ ( column-name, ... ) ]
... REFERENCES table-name [ ( column-name, ... ) ]
```

... [ *actions* ] [ **CHECK ON COMMIT** ]

*actions:*

[ **ON UPDATE** *action* ] [ **ON DELETE** *action* ]

*action:*

**CASCADE**  
**| SET NULL**  
**| SET DEFAULT**  
**| RESTRICT**

**Usage**

Anywhere.

**Permissions**

Must be the **owner** of the table or have DBA authority. Requires exclusive access to the table.

**Side effects**

Automatic commit. The **MODIFY** and **DELETE** options close all cursors for the current connection. The ISQL data window is also cleared.

**See also**

CREATE TABLE statement  
DROP statement  
"SQL Anywhere Data Types"

**Description**

The ALTER TABLE statement changes table attributes (column definitions, constraints) in a table that was previously created. Note that the syntax allows a list of alter clauses; however, only one table-constraint or column-constraint can be added, modified or deleted in one ALTER TABLE statement.

**ADD column-definition** Add a new column to the table. The table must be empty to specify NOT NULL.

**NULL values**

SQL Anywhere optimizes the creation of columns which are allowed to contain the NULL value. The first column that is allowed to contain the NULL value allocates room for eight such columns, and initializes all eight to be the NULL value. (This requires no extra storage.) Thus, the next seven columns added require no changes to the rows of the table. Adding one more column will then allocate room for another eight such columns and then modify each row of the table to allocate the extra space. Consequently, seven out of eight column additions run quickly.

**ADD table-constraint** Add a constraint to the table. See "CREATE TABLE statement" on page 1020 for a full explanation of table constraints.

If PRIMARY KEY is specified, the table must not already have a primary key created by the CREATE TABLE statement or another ALTER TABLE statement.

**MODIFY column-definition** Change the length or data type of an existing column in a table. If NOT NULL is specified, a NOT NULL constraint is added to the named column. Otherwise, the NOT NULL constraint for the column will not be changed. If necessary, the data in the modified column will be converted to the new data type. If a conversion error occurs, the operation will fail and the table will be left unchanged.

**Deleting an index, constraint, or key**

If the column is contained in a uniqueness constraint, a foreign key, or a primary key then the constraint or key must be deleted before the column can be modified. If a primary key is deleted, all foreign keys referencing the table will also be deleted.

You cannot MODIFY a table or column constraint. To change a constraint, you must DELETE the old constraint and ADD the new constraint.

**MODIFY column-name DEFAULT default-value** Change the default value of an existing column in a table. To remove a default value for a column, specify DEFAULT NULL.

**MODIFY column-name [ NOT ] NULL** Change the NOT NULL constraint on the column to allow or disallow NULL values in the column.

**MODIFY column-name CHECK NULL** Delete the check constraint for the column. This statement cannot be used on databases created before Release 5.0.

**MODIFY column-name CHECK ( condition )** Replace the existing CHECK condition for the column with the one specified. This statement cannot be used on databases created before SQL Anywhere 5.0.

**DELETE column-name** Delete the column from the table. If the column is contained in any index, uniqueness constraint, foreign key, or primary key then the index, constraint or key must be deleted before the column can be deleted. This does not delete CHECK constraints that refer to the column.

**DELETE CHECK** Delete all check constraints for the table. This includes both table check constraints and column check constraints.

**DELETE UNIQUE (column-name,...)** Delete a uniqueness constraint for this table. Any foreign keys referencing this uniqueness constraint (rather than the primary key) will also be deleted.

**DELETE PRIMARY KEY** Delete the primary key constraint for this table. All foreign keys referencing the primary key for this table will also be deleted.

**DELETE FOREIGN KEY role-name** Delete the foreign key constraint for this table with the given role name.

**RENAME new-table-name** Change the name of the table to the *new-table-name*. Note that any applications using the old table name will need to be modified. Also, any foreign keys which were automatically assigned the same name as the old table name will not change names.

**RENAME column-name TO new-column-name** Change the name of the column to the *new-column-name*. Note that any applications using the old column name will need to be modified.

**ALTER TABLE** will be prevented whenever the statement affects a table that is currently being used by another connection. **ALTER TABLE** can be time consuming and the server will not process requests referencing the same table while the statement is being processed.

Before Version 5.0, all table and column constraints were held in a single table constraint. Consequently, for these databases individual constraints on columns cannot be deleted using the **MODIFY column-name CHECK NULL** clause or replaced using the **MODIFY column-name CHECK (condition)** clause. To use these statements, the entire table constraint should be deleted and the constraints added back using the **MODIFY column-name CHECK (condition)** clause.

## Examples

Add a new column to the employees table showing which office they work in.

```
ALTER TABLE employee
ADD office CHAR(20) DEFAULT 'Boston'
```

Drop the office column from the employees table.

```
ALTER TABLE employee
DELETE office
```

The address column in the customer table can currently hold up to 35 characters. Allow it to hold up to 50 characters.

```
ALTER TABLE customer
MODIFY address CHAR(50)
```

Add a column to the customer table assigning each customer a sales contact.

```
ALTER TABLE customer
ADD sales_contact INTEGER
REFERENCES employee (emp_id)
ON UPDATE CASCADE
```



ON DELETE SET NULL

This foreign key is constructed with a cascading updates and is set null on deletes. If an employee has their employee ID changed, the column is updated to reflect this change. If an employee leaves the company and has their employee ID deleted, the column is set to NULL.

## ALTER TRIGGER statement

<b>Function</b>	To replace a trigger definition with a modified version. You must include the entire new trigger definition in the ALTER TRIGGER statement.
<b>Syntax</b>	<pre><b>ALTER TRIGGER</b> <i>trigger-name</i> <i>trigger-time</i> <i>trigger-event</i> [, <i>trigger-event</i>,...]     ... [ <b>ORDER</b> <i>integer</i> ] <b>ON</b> <i>table-name</i>     ... [ <b>REFERENCING</b> [ <b>OLD AS</b> <i>old-name</i> ]         [ <b>NEW AS</b> <i>new-name</i> ] ]         [ <b>REMOTE AS</b> <i>remote-name</i> ] ]     ... [ <b>FOR EACH</b> { <b>ROW</b>   <b>STATEMENT</b> } ]     ... [ <b>WHEN</b> ( <i>search-condition</i> ) ]     ... [ <b>IF UPDATE</b> ( <i>column-name</i> ) <b>THEN</b>     ... [ { <b>AND</b>   <b>OR</b> } <b>UPDATE</b> ( <i>column-name</i> ) ] ... ]         ... <i>compound-statement</i>     ... [ <b>ELSEIF UPDATE</b> ( <i>column-name</i> ) <b>THEN</b>     ... [ { <b>AND</b>   <b>OR</b> } <b>UPDATE</b> ( <i>column-name</i> ) ] ... ]         ... <i>compound-statement</i>     ... <b>END IF</b> ] ]</pre>
<b>Parameters</b>	<p><i>trigger-time</i>: <b>BEFORE</b>   <b>AFTER</b>   <b>RESOLVE</b></p> <p><i>trigger-event</i>: <b>DELETE</b>   <b>INSERT</b>   <b>UPDATE</b>   <b>UPDATE OF</b> <i>column-list</i></p>
<b>Usage</b>	Anywhere.
<b>Permissions</b>	Must be the owner of the table on which the trigger is defined, or be DBA.
<b>Side effects</b>	Automatic commit.
<b>See also</b>	CREATE TRIGGER statement DROP statement
<b>Description</b>	The ALTER TRIGGER statement is identical in syntax to the CREATE TRIGGER statement except for the first word. The ALTER TRIGGER statement replaces the entire contents of the CREATE TRIGGER statement with the contents of the ALTER TRIGGER statement.

## ALTER VIEW statement

<b>Function</b>	To replace a view definition with a modified version. You must include the entire new view definition in the ALTER VIEW statement, and reassign permissions on the view
<b>Syntax</b>	<b>ALTER VIEW</b> ... [ <i>owner</i> .] <i>view-name</i> [( <i>column-name</i> , ... )] ... <b>AS</b> <i>select-without-order-by</i> ... [ <b>WITH CHECK OPTION</b> ]
<b>Usage</b>	Anywhere.
<b>Permissions</b>	Must be owner of the view or have DBA authority.
<b>Side effects</b>	Automatic commit.
<b>See also</b>	CREATE VIEW statement DROP statement
<b>Description</b>	The ALTER VIEW statement is identical in syntax to the CREATE VIEW statement except for the first word. The ALTER VIEW statement replaces the entire contents of the CREATE VIEW statement with the contents of the ALTER VIEW statement. Existing permissions on the view are maintained, and do not have to be reassigned. If a DROP VIEW and CREATE VIEW were carried out, permissions on the view would have to be reassigned.

# CALL statement

<b>Function</b>	To invoke a procedure.
<b>Syntax</b>	[ <i>variable</i> = ] <b>CALL</b> <i>procedure-name</i> ( [ <i>expression</i> ,... ] ) [ <i>variable</i> = ] <b>CALL</b> <i>procedure-name</i> ( [ <i>parameter-name</i> = <i>expression</i> ,... ] )
<b>Usage</b>	Anywhere.
<b>Permissions</b>	Must be the owner of the procedure, have EXECUTE permission for the procedure, or have DBA authority.
<b>Side effects</b>	None.
<b>See also</b>	CREATE PROCEDURE statement GRANT statement The chapter "Using Procedures, Triggers, and Batches"
<b>Description</b>	<p>The CALL statement invokes a procedure that has been previously created with a CREATE PROCEDURE statement. When the procedure completes, any INOUT or OUT parameter values will be copied back.</p> <p>The argument list can be specified by position or by using keyword format. By position, the arguments will match up with the corresponding parameter in the parameter list for the procedure. By keyword, the arguments are matched up with the named parameters.</p> <p>All arguments are optional: procedure arguments can be assigned default values in the CREATE PROCEDURE statement, and missing parameters are assigned the default value or, if no default is set, the parameter is set to NULL.</p> <p>Inside a procedure, a CALL statement can be used in a DECLARE statement when the procedure returns result sets (see "Returning results from procedures" in the chapter "Using Procedures, Triggers, and Batches").</p> <p>Procedures can return a value (as a status indicator, say) using the RETURN statement. You can save this return value in a variable using using the quality sign as an assignment operator:</p> <pre>CREATE VARIABLE returnval INT ; returnval = CALL proc_integer ( arg1 = vall, ... )</pre>
<b>Examples</b>	<p>Call the sp_customer_list procedure. This procedure has no parameters, and returns a result set.</p> <pre>CALL sp_customer_list()</pre>

The following ISQL example creates a procedure to return the number of orders placed by the customer whose ID is supplied, creates a variable to hold the result, calls the procedure, and displays the result.

```
% Set the statement delimiter to create the
procedure
SET OPTION COMMAND_DELIMITER = ';;'
% Create the procedure
CREATE PROCEDURE OrderCount (IN customer_ID INT, OUT
Orders INT)
BEGIN
SELECT COUNT("DBA".sales_order.id)
INTO Orders
FROM "DBA".customer
KEY LEFT OUTER JOIN "DBA".sales_order
WHERE "DBA".customer.id = customer_ID ;
END ;;
% Reset the statement delimiter to semicolon.
SET OPTION COMMAND_DELIMITER = ';'
% Create a variable to hold the result
CREATE VARIABLE Orders INT ;
% Call the procedure, FOR customer 101
%-----
CALL OrderCount ( 101, Orders) ;
%-----
% Display the result
SELECT Orders FROM DUMMY ;
```

# CASE statement

<b>Function</b>	Select execution path based on multiple cases.
<b>Syntax</b>	<pre><b>CASE</b> <i>value-expression</i> ... <b>WHEN</b> [ <i>constant</i>   <b>NULL</b> ] <b>THEN</b> <i>statement-list</i> ... ... [ <b>WHEN</b> [ <i>constant</i>   <b>NULL</b> ] <b>THEN</b> <i>statement-list</i> ] ... ... <b>ELSE</b> <i>statement-list</i> ... <b>END CASE</b></pre>
<b>Usage</b>	Procedures, triggers, and batches.
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>See also</b>	Compound statements The chapter "Using Procedures, Triggers, and Batches"
<b>Description</b>	The CASE statement is a control statement that allows you to choose a list of SQL statements to execute based on the value of an expression. If a WHEN clause exists for the value of <i>value-expression</i> , the <i>statement-list</i> in the WHEN clause is executed. If no appropriate WHEN clause exists, and an ELSE clause exists, the <i>statement-list</i> in the ELSE clause is executed. Execution resumes at the first statement after the END CASE.
<b>Example</b>	<p>The following procedure using a case statement classifies the products listed in the product table of the sample database into one of shirt, hat, shorts, or unknown.</p> <p>The following procedure uses a case statement to classify the results of a query.</p>

```
CREATE PROCEDURE ProductType (IN product_id INT, OUT
type CHAR(10))
BEGIN
  DECLARE prod_name CHAR(20) ;
  SELECT name INTO prod_name FROM "DBA"."product"
  WHERE id = product_id;
  CASE prod_name
  WHEN 'Tee Shirt' THEN
    SET type = 'Shirt'
  WHEN 'Sweatshirt' THEN
    SET type = 'Shirt'
  WHEN 'Baseball Cap' THEN
    SET type = 'Hat'
  WHEN 'Visor' THEN
    SET type = 'Hat'
  WHEN 'Shorts' THEN
    SET type = 'Shorts'
  ELSE
```

```
        SET type = 'UNKNOWN'  
    END CASE ;  
END
```

## **CHECKPOINT statement**

<b>Function</b>	To <b>checkpoint</b> the database.
<b>Syntax</b>	<b>CHECKPOINT</b>
<b>Usage</b>	Anywhere.
<b>Permissions</b>	Must have DBA authority to CHECKPOINT when you are running with multiple users over a LAN. For single-user, no authorization is required.
<b>Side effects</b>	None.
<b>Description</b>	The CHECKPOINT statement will checkpoint the database. Checkpoints are also performed automatically by the database engine. It is not normally required for an application to ever issue the CHECKPOINT statement. For a full description of checkpoints, see "Backup and Data Recovery".



# CLOSE statement

<b>Function</b>	To close a cursor.
<b>Syntax</b>	<b>CLOSE</b> <i>cursor-name</i>
<b>Parameters</b>	<i>cursor-name</i> : <i>identifier</i> , or <i>host-variable</i>
<b>Usage</b>	Embedded SQL, procedures, triggers, and batches. The <b>host-variable</b> format is for Embedded SQL only.
<b>Permissions</b>	The cursor must have been previously opened.
<b>Side effects</b>	None.
<b>See also</b>	OPEN statement DECLARE CURSOR statement PREPARE statement

**Description** This statement closes the named cursor.

**Examples** The following examples close cursors in Embedded SQL.

```
EXEC SQL CLOSE employee_cursor;
EXEC SQL CLOSE :cursor_var;
```

The following procedure uses a cursor.

```
CREATE PROCEDURE TopCustomer (OUT TopCompany
CHAR(35), OUT TopValue INT)
BEGIN
  DECLARE err_notfound EXCEPTION
    FOR SQLSTATE '02000' ;
  DECLARE curThisCust CURSOR FOR
  SELECT company_name, CAST(
  sum(sales_order_items.quantity *
  product.unit_price) AS INTEGER) VALUE
  FROM customer
  LEFT OUTER JOIN sales_order
  LEFT OUTER JOIN sales_order_items
  LEFT OUTER JOIN product
  GROUP BY company_name ;

  DECLARE ThisValue INT ;
  DECLARE ThisCompany CHAR(35) ;
  SET TopValue = 0 ;
  OPEN curThisCust ;
  CustomerLoop:
  LOOP
    FETCH NEXT curThisCust
    INTO ThisCompany, ThisValue ;
    IF SQLSTATE = err_notfound THEN
```

## *CLOSE statement*

---

```
        LEAVE CustomerLoop ;
    END IF ;
    IF ThisValue > TopValue THEN
        SET TopValue = ThisValue ;
        SET TopCompany = ThisCompany ;
    END IF ;
    END LOOP CustomerLoop ;
    CLOSE curThisCust ;
END
```

# COMMENT statement

<b>Function</b>	To store a comment in the system tables for a database object.
<b>Syntax</b>	<pre>COMMENT ON {     COLUMN [ owner.]table-name.column-name     FOREIGN KEY [ owner.]table-name.role-name     INDEX [ owner.]index-name     PUBLICATION [ owner.]publication-name     PROCEDURE [ owner.]procedure-name     SUBSCRIPTION [ owner.]subscription-name     TABLE [ owner.]table-name     USER userid     TRIGGER [ owner.]trigger-name     VIEW [ owner.]view-name } IS comment</pre>
<b>Parameters</b>	<p><i>comment:</i>  { <i>string</i>   <b>NULL</b> }</p>
<b>Usage</b>	Anywhere.
<b>Permissions</b>	Must either be the owner of the database object being commented, or have DBA authority.
<b>Side effects</b>	Automatic commit.
<b>Description</b>	<p>Several system tables have a column named Remarks that allows you to associate a comment with a database item (SYSUSERPERM, SYSTABLE, SYSCOLUMN, SYSINDEX, SYSFOREIGNKEY, SYSPROCEDURE, SYSTRIGGER). The COMMENT ON statement allows you to set the Remarks column in these system tables. A comment can be removed by setting it to NULL.</p> <p>When adding a comment to an index or trigger, the owner is the owner of the table on which the index or trigger is defined.</p>
<b>Examples</b>	<p>The following examples show how to add and remove a comment.</p> <ul style="list-style-type: none"> <li>◆ Add a comment to the employee table. <pre>COMMENT ON TABLE employee IS "Employee information"</pre> </li> <li>◆ Remove the comment from the employee table. <pre>COMMENT ON TABLE employee</pre> </li> </ul>

IS NULL

# COMMIT statement

<b>Function</b>	To make any changes to the database permanent.
<b>Syntax</b>	<b>COMMIT [ WORK ]</b>
<b>Usage</b>	Anywhere.
<b>Permissions</b>	Must be connected to the database.
<b>Side effects</b>	Closes all cursors that were not opened with the WITH HOLD option.
<b>See also</b>	ROLLBACK statement PREPARE TO COMMIT statement CONNECT statement SET CONNECTION statement DISCONNECT statement
<b>Description</b>	<p>The COMMIT statement ends a logical unit of work (transaction) and makes all changes made during this transaction permanent in the database. A transaction is defined as the database work done between successful COMMIT and ROLLBACK statements on a single database connection.</p> <p>The COMMIT statement is also used as the second phase of a two-phase commit operation. See "Coordinating transactions with multiple database engines" in the chapter "Using Transactions and Locks" and "PREPARE TO COMMIT statement" on page 1123 for more information.</p> <p>The changes committed are those made by the data manipulation statements: INSERT, UPDATE, and DELETE, as well as the ISQL load statement INPUT.</p> <p>The data definition statements all do an automatic commit. They are:</p> <ul style="list-style-type: none"><li>◆ ALTER</li><li>◆ COMMENT</li><li>◆ CREATE</li><li>◆ DROP</li><li>◆ GRANT</li><li>◆ REVOKE</li><li>◆ SET OPTION</li></ul>

The COMMIT statement fails if SQL Anywhere detects any invalid foreign keys. This makes it impossible to end a transaction with any invalid foreign keys. Usually, foreign key integrity is checked on each data manipulation operation. However, if either the database option WAIT\_FOR\_COMMIT is set ON or a particular foreign key was defined with a CHECK ON COMMIT clause, the database engine will not check integrity until the COMMIT statement is executed. For a two-phase commit operation, these errors will be reported on the first phase (PREPARE TO COMMIT), not on the second phase (COMMIT).

# Compound statements

<b>Function</b>	To specify a statement that groups other statements together.
<b>Syntax</b>	<pre>[ <i>statement-label</i> : ] ... <b>BEGIN</b> [ [ <b>NOT</b> ] <b>ATOMIC</b> ] ... [ <i>local-declaration</i> ; ... ] ... <i>statement-list</i> ... [ <b>EXCEPTION</b> [ <i>exception-case</i> ... ] ] ... <b>END</b> [ <i>statement-label</i> ]</pre>
<b>Parameters</b>	<p><i>local-declaration</i>:</p> <pre><i>variable-declaration</i>   <i>cursor-declaration</i>   <i>exception-declaration</i>   <i>temporary-table-declaration</i></pre> <p><i>variable-declaration</i>:</p> <pre><b>DECLARE</b> <i>variable-name</i> <i>data-type</i></pre> <p><i>exception-declaration</i>:</p> <pre><b>DECLARE</b> <i>exception-name</i> <b>EXCEPTION</b> <b>FOR SQLSTATE</b> [ <b>VALUE</b> ] <i>string</i></pre> <p><i>exception-case</i>:</p> <pre><b>WHEN</b> <i>exception-name</i> [ ,... ] <b>THEN</b> <i>statement-list</i>   <b>WHEN OTHERS</b> <b>THEN</b> <i>statement-list</i></pre>
<b>Usage</b>	Procedures, triggers, and batches.
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>See also</b>	<p>DECLARE CURSOR statement</p> <p>DECLARE TEMPORARY TABLE statement</p> <p>LEAVE statement</p> <p>SIGNAL statement</p> <p>RESIGNAL statement</p> <p>The chapter "Using Procedures, Triggers, and Batches"</p>
<b>Description</b>	The body of a procedure or trigger is a <b>compound statement</b> . Compound statements can also be used in control statements within a procedure or trigger.

A compound statement allows one or more SQL statements to be grouped together and treated as a unit. A compound statement starts with the keyword `BEGIN` and ends with the keyword `END`. Immediately following the `BEGIN`, a compound statement can have local declarations that only exist within the compound statement. A compound statement can have a local declaration for a variable, a cursor, a temporary table, or an exception. Local declarations can be referenced by any statement in that compound statement, or in any compound statement nested within it. Local declarations are not visible to other procedures that are called from within a compound statement.

If the ending *statement-label* is specified, it must match the beginning *statement-label*. The `LEAVE` statement can be used to resume execution at the first statement after the compound statement. The compound statement that is the body of a procedure or triggers has an implicit label that is the same as the name of the procedure or trigger.

☞ For a complete description of compound statements and exception handling, see the chapter "Using Procedures, Triggers, and Batches".

### Example

The body of a procedure or trigger is a compound statement.

```
CREATE PROCEDURE TopCustomer (OUT TopCompany
CHAR(35), OUT TopValue INT)
BEGIN
    DECLARE err_notfound EXCEPTION FOR
        SQLSTATE '02000' ;
    DECLARE curThisCust CURSOR FOR
        SELECT company_name, CAST(
            sum(sales_order_items.quantity *
            product.unit_price) AS INTEGER) VALUE
        FROM customer
            LEFT OUTER JOIN sales_order
            LEFT OUTER JOIN sales_order_items
            LEFT OUTER JOIN product
        GROUP BY company_name ;
    DECLARE ThisValue INT ;
    DECLARE ThisCompany CHAR(35) ;
    SET TopValue = 0 ;
    OPEN curThisCust ;

    CustomerLoop:
    LOOP
        FETCH NEXT curThisCust
            INTO ThisCompany, ThisValue ;
        IF SQLSTATE = err_notfound THEN
            LEAVE CustomerLoop ;
        END IF ;
        IF ThisValue > TopValue THEN
            SET TopValue = ThisValue ;
            SET TopCompany = ThisCompany ;
        END IF ;
    END LOOP ;
END ;
```



```
END LOOP CustomerLoop ;  
CLOSE curThisCust ;  
END
```

## **CONFIGURE statement**

**Function** To activate the ISQL configuration window.

**Syntax** **CONFIGURE**

**Usage** ISQL.

**Permissions** None.

**Side effects** None.

**See also** SET OPTION statement

**Description** The CONFIGURE statement activates the ISQL configuration window. This window displays the current settings of all ISQL options. It does not display or allow you to modify database options or options for other software that are stored in the database. The SET OPTIONS statement (see "SET OPTION statement" on page 1149) allows you to set all options—ISQL, database and other options.

The TAB key can be used to change fields. If a hardware cursor appears in a field, then you are allowed to type a value for the option. If the hardware cursor does not appear, you can use the cursor up and down keys to select one of the allowed values. When you have changed the options, press ENTER to save the changes and exit, or ESCAPE to undo the changes and exit.

If you press ENTER and you have selected Save Options to Database then the options will be written to the SYSOPTION table in the database and the database engine will perform an automatic COMMIT. If you do not select Save Options to Database, then the options are set temporarily and remain in effect for the current database connection only.

# CONNECT statement

<b>Function</b>	To establish a connection to a database.
<b>Syntax</b>	<p>Syntax 1</p> <pre>CONNECT [ TO <i>engine-name</i> ]         ... [ DATABASE <i>database-name</i> ]         ... [ AS <i>connection-name</i> ]         ... [ USER ] <i>userid</i> [ IDENTIFIED BY <i>password</i> ]</pre> <p>Syntax 2</p> <pre>CONNECT USING <i>connect-string</i></pre>
<b>Parameters</b>	<p><i>engine-name</i>: <i>identifier, string or host-variable</i></p> <p><i>database-name</i>: <i>identifier, string or host-variable</i></p> <p><i>connection-name</i>: <i>identifier, string or host-variable</i></p> <p><i>userid</i>: <i>identifier, string or host-variable</i></p> <p><i>password</i>: <i>identifier, string or host-variable</i></p> <p><i>connect-string</i>: <i>a valid connection string or host variable</i></p>
<b>Usage</b>	ISQL and Embedded SQL
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>See also</b>	<p>GRANT statement</p> <p>DISCONNECT statement</p> <p>SET CONNECTION statement</p>
<b>Description</b>	<p>The CONNECT statement establishes a connection to the database identified by <i>database-name</i> running on the engine or server identified by <i>engine-name</i>.</p> <p><b>Embedded SQL behavior</b> In Embedded SQL, if no <i>engine-name</i> is specified, the default local database engine will be assumed (the first database engine started). If a local database engine is not running and the SQL Anywhere Client (DBCLIENT) is running, the default server will be assumed (the server name specified when the client was started). If no <i>database-name</i> is specified, the first database on the given engine or server will be assumed.</p>

**ISQL behavior** If no database or engine is specified in the **CONNECT** statement, ISQL remains connected to the current database, rather than to the default engine and database. If a database name is specified without an engine name, ISQL will try to connect to the specified database on the current engine. If an engine name is specified without a database name, ISQL will connect to the default database on the specified engine. For example, in the following batch, the two tables are created in the same database.

```
CREATE TABLE t1( c1 int );
CONNECT DBA IDENTIFIED BY SQL;
CREATE TABLE t2 (c1 int );
```

No other database statements are allowed until a successful **CONNECT** statement has been executed.

(In Embedded SQL, **WHENEVER**, **SET SQLCA** and some **DECLARE** statements do not generate code and thus may appear before the **CONNECT** statement in the source file.)

The user ID and password are used for checking the permissions on SQL statements. If the password or the user ID and password are not specified, the user will be prompted to type the missing information.

In Embedded SQL, the user ID and password are used for permission checks on all dynamic SQL statements. Static SQL statements use the user ID and password specified with the *-l* option on the SQLPP statement line. If no *-l* option is given, then the user ID and password of the **CONNECT** statement are used for static SQL statements also.

If you are connected to a user ID with DBA authority, you can connect to another user ID without specifying a password. (The output of DBTRAN requires this capability.) For example, if you are connected to a database as DBA, you can connect without a password with the statement:

```
connect other_user_id
```

In Embedded SQL, you can connect without a password by using a host variable for the password and setting the value of the host variable to be the null pointer.

A connection can optionally be named by specifying the **AS** clause. This allows multiple connections to the same database, or multiple connections to the same or different database servers, all simultaneously. Each connection has its own associated transaction. You may even get locking conflicts between your transactions if, for example, you try to modify the same record in the same database from two different connections.

Multiple connections are managed through the concept of a current connection. After a successful connect statement, the new connection becomes the current one. To switch to a different connection, use the SET CONNECTION statement. The DISCONNECT statement is used to drop connections.

The dictionary clause allows you to specify the directory that contains the data dictionary you want to log in to (files \*.DFF).

A database directory can also be specified. This specifies the directory or list of directories that contain the data files you will be accessing in the upcoming session (files \*.DAT).

For Syntax 2, a **connect string** is a list of parameter settings of the form **keyword=value**. For a description of valid settings for the connection string, see "Database connection parameters" in the chapter "Connecting to a Database". The connect string must be enclosed in single quotes.

## Examples

- ◆ The following are examples of CONNECT usage within Embedded SQL.

```
EXEC SQL CONNECT AS :conn_name
USER :userid IDENTIFIED BY :password;

EXEC SQL CONNECT USER "dba" IDENTIFIED BY "sql";
```

- ◆ The following are examples of CONNECT usage from ISQL.

- ◆ Connect to a database from ISQL. ISQL will prompt for a user ID and a password.

```
CONNECT
```

- ◆ Connect to the default database as DBA, from ISQL. ISQL will prompt for a password.

```
CONNECT USER "DBA"
```

- ◆ Connect to the sample database as the DBA, from ISQL.

```
CONNECT
TO sademo
USER "DBA"
IDENTIFIED BY sql
```

- ◆ Connect to the sample database using a connect string, from ISQL.

```
CONNECT
USING 'UID=DBA;PWD=sql;DBN=sademo'
```

## **CREATE DATATYPE statement**

<b>Function</b>	To create a user-defined data type in the database.
<b>Syntax</b>	<pre><b>CREATE</b> { <b>DATATYPE</b>   <b>DOMAIN</b> } [ <b>AS</b> ] <i>domain-name data-type</i>     ... [ [ <b>NOT</b> ] <b>NULL</b> ]     ... [ <b>DEFAULT</b> <i>default-value</i> ]     ... [ <b>CHECK</b> ( <i>condition</i> ) ]</pre>
<b>Parameters</b>	<p><i>domain-name:</i>     <i>identifier</i></p> <p><i>data-type:</i>     <i>built-in data type, with precision and scale</i></p>
<b>Usage</b>	Anywhere.
<b>Permissions</b>	Must have RESOURCE authority.
<b>Side effects</b>	Automatic commit.
<b>See also</b>	DROP statement "SQL Anywhere Data Types"
<b>Description</b>	<p>User-defined data types are aliases for built-in data types, including precision and scale values where applicable. They improve convenience and encourage consistency in the database.</p> <p>It is recommended that you use CREATE DOMAIN, rather than CREATE DATATYPE, as CREATE DOMAIN is the ANSI/ISO SQL3 term.</p> <p>User-defined data types can have CHECK conditions and DEFAULT conditions associated with them, and you can indicate whether the data type permits NULL values or not. These conditions are inherited by any column defined on the data type. Any conditions explicitly specified on the column override the data type conditions.</p> <p>The user who creates a data type is automatically made the owner of that data type. No owner can be specified in the CREATE DATATYPE statement. The user-defined data type name must be unique, and all users can access the data type without using the owner as prefix.</p> <p>User-defined data types are objects within the database. Their names must conform to the rules for identifiers. User-defined data type names are always case insensitive, as are built-in data type names.</p>

By default, user-defined data types allow NULLs unless the **allow\_nulls\_by\_default** option is set to OFF. In this case, new user-defined data types by default do not allow NULLs. Any columns created on a user-defined data type either allow or do not allow NULLs depending on the setting of the user-defined data type at the time it was created, not on the current setting of the **allow\_nulls\_by\_default** option. Any explicit setting of NULL or NOT NULL in the column definition overrides the user-defined data type setting.

When creating a CHECK condition, you can use a variable name prefixed with the @ sign in the condition. When the data type is used in the definition of a column, such a variable is replaced by the column name. This allows CHECK conditions to be defined on data types and used by columns of any name.

To drop the data type from the database, use the DROP statement. You must be either the owner of the data type or have DBA authority in order to drop a user-defined data type.

**Example**

The following statement creates a data type named address, which holds a 35-character string, and which may be NULL.

```
CREATE DATATYPE address CHAR( 35 ) NULL
```

The following statement creates a data type named id, which does not allow NULLS, and which is autoincremented by default.

```
CREATE DATATYPE id INT  
NOT NULL  
DEFAULT AUTOINCREMENT
```

## CREATE DBSPACE statement

<b>Function</b>	To create a new database file. This file may be on a different device.
<b>Syntax</b>	<b>CREATE DBSPACE</b> <i>dbspace-name</i> <b>AS</b> <i>filename</i>
<b>Usage</b>	Anywhere.
<b>Permissions</b>	Must have DBA authority.
<b>Side effects</b>	Automatic commit.
<b>See also</b>	DROP statement
<b>Description</b>	<p>The CREATE DBSPACE statement creates a new database file. When a database is first initialized using DBINIT, it is composed of one file. All tables and indexes created are placed in that file. CREATE DBSPACE adds a new file to the database. This file can be on a different disk drive than the root file allowing the creation of databases larger than one physical device.</p> <p>The <i>dbspace-name</i> is an internal name for the database file. The <i>filename</i> is the actual name of the database file, with a path where necessary.</p> <p>A <i>filename</i> without an explicit directory is created in the same directory as the main database file. Any relative directory is relative to the main database file. When you are using a SQL Anywhere, the <i>filename</i> is a filename on the server machine. When you are using the SQL Anywhere Server for NetWare, the <i>filename</i> should use a volume name (not a drive letter) when an absolute directory is specified.</p> <p>Each table, including its associated indexes, is contained entirely within one database file. The IN clause of the CREATE TABLE statement the <i>dbspace</i> into which a table is placed. Tables are put into the root database file by default.</p>
<b>Example</b>	<p>Create a <i>dbspace</i> called <i>library</i> to hold the <i>LibraryBooks</i> table and its indices.</p>

```
CREATE DBSPACE library
AS 'e:\dbfiles\library.db' ;

CREATE TABLE LibraryBooks (
title char(100),
author char(50),
isbn char(30),
) IN library ;
```



# CREATE FUNCTION statement

<b>Function</b>	To create a new function in the database.
<b>Syntax</b>	<pre> <b>CREATE FUNCTION</b> [ <i>owner</i>.]<i>function-name</i> ( [ <i>parameter</i> , ... ] )     ... <b>RETURNS</b> <i>data-type</i>     ... { <b>EXTERNAL NAME</b> <i>library-call</i>       ... [ <b>ON EXCEPTION RESUME</b> ]     ... <i>compound-statement</i> } </pre>
<b>Parameters</b>	<p><i>parameter</i>:</p> <p><i>parameter-name data-type</i></p> <p><i>library-call</i>:</p> <p>'[<i>operating-system</i>.]<i>function-name</i>@<i>library.dll</i>; ...'</p> <p><i>operating-system</i>:</p> <p><b>OS2</b></p> <p>  <b>Windows3X</b></p> <p>  <b>Windows95</b></p> <p>  <b>WindowsNT</b></p> <p>  <b>NetWare</b></p>
<b>Usage</b>	Anywhere.
<b>Permissions</b>	Must have RESOURCE authority.
<b>Side effects</b>	Automatic commit.
<b>See also</b>	<p>DROP statement</p> <p>Compound statements</p> <p>CREATE PROCEDURE statement</p> <p>RETURN statement</p> <p>The chapter "Using Procedures, Triggers, and Batches"</p>
<b>Description</b>	<p>The CREATE FUNCTION statement creates (stores) a user-defined function in the database. A function can be created for another user by specifying an owner name. Subject to permissions, a user-defined function can be used in exactly the same way as other nonaggregate functions.</p> <p>Parameter names must conform to the rules for other database identifiers such as column names. They must be one of the types supported by SQL Anywhere (see "SQL Anywhere Data Types"), and must be prefixed by the keyword IN, signifying that the argument is an expression that provides a value to the procedure.</p>

A function using the EXTERNAL NAME clause is a wrapper around a call to an external dynamic link library, and is called an external stored procedure. An external stored procedure can have no clauses other than the EXTERNAL NAME clause following the RETURNS clause. For a description of external procedures, see "Calling external libraries from stored procedures" in the chapter "Using Procedures, Triggers, and Batches".

**Example**

The following function concatenates a firstname string and a lastname string.

```
CREATE FUNCTION fullname ( firstname CHAR(30),
                           lastname CHAR(30) )
RETURNS CHAR(61)
BEGIN
    DECLARE name CHAR(61) ;
    SET name = firstname || ' ' || lastname ;
    RETURN (name) ;
END
```

The following ISQL statements illustrate the use of the fullname function.

Return a full name from two supplied strings:

```
SELECT fullname ('joe', 'smith')
```

---

**fullname('joe','smith')**

joe smith

List the names of all employees:

```
SELECT fullname (emp_fname, emp_lname)
FROM employee
```

---

**fullname (emp\_fname, emp\_lname)**

Fran Whitney

Matthew Cobb

Philip Chin

Julie Jordan

Robert Breault

...

# CREATE INDEX statement

<b>Function</b>	To create an index on a specified table. Indexes are used to improve database performance.
<b>Syntax</b>	<pre>CREATE [ UNIQUE ] INDEX <i>index-name</i> ... ON [ <i>owner.</i>]<i>table-name</i> ... ( <i>column-name</i> [ ASC   DESC ], ... ) ... [ { IN   ON } <i>dbspace-name</i> ]</pre>
<b>Usage</b>	Anywhere.
<b>Permissions</b>	Must be the owner of the table or have DBA authority.
<b>Side effects</b>	Automatic commit.
<b>See also</b>	DROP statement
<b>Description</b>	<p>The CREATE INDEX statement creates a sorted index on the specified columns of the named table. Indexes are automatically used by SQL Anywhere to improve the performance of queries issued to the database as well as for sorting queries where an ORDER BY clause is specified. Once an index is created, it is never referenced again except to delete it using the DROP INDEX statement.</p> <p><b>UNIQUE constraint</b> The UNIQUE constraint ensures that there will not be two rows in the table with identical values in all the columns in the index.</p> <p><b>Ascending or descending sorting</b> Columns are sorted in ascending (increasing) order unless descending (DESC) is explicitly specified. An index will be used by SQL Anywhere for both an ascending and a descending ORDER BY, no matter whether the index was ascending or descending. However, if an ORDER BY is performed with mixed ascending and descending attributes, an index will be used only if the index was created with the same ascending and descending attributes.</p> <p><b>Index placement</b> By default, the index is placed in the same database file as its table. You can place the index in a separate database file by specifying a dbspace name in which to put the index. This feature is of use mainly for large databases, to circumvent the limit of 2 GB per table.</p>
<b>Notes</b>	<ul style="list-style-type: none"> <li>◆ <b>Index ownership</b> There is no way of specifying the index owner in the CREATE INDEX statement. Indexes are automatically owned by the owner of the table on which they are defined. The index name must be unique for each owner.</li> <li>◆ <b>No indexes on views</b> Indexes cannot be created for views.</li> </ul>

- ◆ **Index name space** The name of each index must be unique for a given table. For databases created previous to release 5.5.01, the condition was more restrictive: that each index name must be unique for a given user ID.
- ◆ **Exclusive table use** CREATE INDEX is prevented whenever the statement affects a table currently being used by another connection. CREATE INDEX can be time consuming and the server will not process requests referencing the same table while the statement is being processed.
- ◆ **Automatically created indexes** SQL Anywhere automatically creates indexes for primary keys and for unique constraints. These automatically-created indexes are held in the same database file as the table.

### Examples

- ◆ Create a two-column index on the employee table.

```
CREATE INDEX employee_name_index
ON employee
( emp_lname, emp_fname )
```

- ◆ Create an index on the sales\_order\_items table for the product ID column.

```
CREATE INDEX item_prod
ON sales_order_items
( prod_id )
```

# CREATE PROCEDURE statement

<b>Function</b>	To create a new procedure in the database.
<b>Syntax</b>	<pre> <b>CREATE PROCEDURE</b> [ <i>owner</i>.]<i>procedure-name</i> ( [ <i>parameter</i> , ... ] ) ... {   [ <b>RESULT</b> ( <i>result-column</i> , ... ) ] [ <b>ON EXCEPTION RESUME</b> ] ... <i>compound-statement</i>     <b>EXTERNAL NAME</b> <i>library-call</i> } </pre>
<b>Parameters</b>	<p><i>parameter</i>:</p> <pre> <i>parameter_mode</i> <i>parameter-name</i> <i>data-type</i> [ <b>DEFAULT</b> <i>expression</i> ]   <b>SQLCODE</b>   <b>SQLSTATE</b> </pre> <p><i>parameter_mode</i>:</p> <pre> <b>IN</b>   <b>OUT</b>   <b>INOUT</b> </pre> <p><i>result-column</i>:</p> <pre> <i>column-name</i> <i>data-type</i> </pre> <p><i>library-call</i>:</p> <pre> '[<i>operating-system</i>.]<i>function-name</i>@<i>library.dll</i>; ...' </pre> <p><i>operating-system</i>:</p> <pre> <b>OS2</b>   <b>Windows3X</b>   <b>Windows95</b>   <b>WindowsNT</b>   <b>NetWare</b> </pre>
<b>Usage</b>	Anywhere.
<b>Permissions</b>	Must have RESOURCE authority.
<b>Side effects</b>	Automatic commit.
<b>See also</b>	<p>DROP statement</p> <p>CALL statement</p> <p>Compound statements</p> <p>GRANT statement</p> <p>CREATE FUNCTION statement</p> <p>EXECUTE IMMEDIATE statement</p> <p>The chapter "Using Procedures, Triggers, and Batches"</p>
<b>Description</b>	The CREATE PROCEDURE statement creates (stores) a procedure in the database. A procedure can be created for another user by specifying a <b>owner</b> . A procedure is invoked with a CALL statement

☞ The body of a procedure consists of a compound statement. For information about compound statements, see "Compound statements" on page 995.

Parameter names must conform to the rules for other database identifiers such as column names. They must be one of the types supported by SQL Anywhere (see "SQL Anywhere Data Types".), and must be prefixed by one of the keywords IN, OUT or INOUT. The keywords have the following meanings:

- ◆ **IN** argument is an expression that provides a value to the procedure.
- ◆ **OUT** argument is a variable that could be given a value by the procedure.
- ◆ **INOUT** argument is a variable that provides a value to the procedure, and could be given a new value by the procedure.

When procedures are executed using the CALL statement, not all parameters need to be specified. If a default value is provided in the CREATE PROCEDURE statement, missing parameters are assigned the default values. If no default value is supplied, the parameter is NULL. SQLSTATE and SQLCODE are special parameters that output the SQLSTATE or SQLCODE value when the procedure ends (they are OUT parameters). Whether or not a SQLSTATE and SQLCODE parameter is specified, the SQLSTATE and SQLCODE special constants can always be checked immediately after a procedure call to test the return status of the procedure. However, the SQLSTATE and SQLCODE special constant values are modified by the next SQL statement. Providing SQLSTATE or SQLCODE as procedure arguments allows the return code to be stored in a variable.

### Result sets

A procedure that returns result sets ("Returning results from procedures" in the chapter "Using Procedures, Triggers, and Batches") may have a RESULT clause. The parenthesized list following the RESULT keyword defines the number of result columns and name and type. This information is returned by the Embedded SQL DESCRIBE or by ODBC SQLDescribeCol when a CALL statement is being described. Allowable data types are listed in "SQL Anywhere Data Types".

Some procedures can return different result sets, with different numbers of columns, depending on how they are executed. For example, the following procedure returns two columns under some circumstances, and one in others.

```
CREATE PROCEDURE names( IN formal char(1))
BEGIN
    IF formal = 'n' THEN
```

```

        SELECT emp_fname
        FROM employee
    ELSE
        SELECT emp_lname, emp_fname
        FROM employee
    END IF
END

```

These variable result set procedures must be written without a **RESULT** clause, or in Transact-SQL. Their use is subject to the following limitations:

- ◆ **Embedded SQL** You must **DESCRIBE** the procedure call after the cursor for the result set is opened, but before any rows are returned, in order to get the proper shape of result set.
- ◆ **ODBC** Variable result set procedures can be used by ODBC applications. The proper description of the variable result sets is carried out by the ODBC driver.
- ◆ **Open Client applications** Variable result set procedures can be used by Open Client applications using the Open Server Gateway. The proper description of the variable result sets is carried out by the Open Server Gateway.
- ◆ **ISQL** ISQL does not support calling of variable result set procedures, and so cannot be used for testing this feature.

If your procedure does not return variable result sets, you should use a **RESULT** clause. The presence of this clause prevents ODBC and Open Client applications from re-describing the result set after a cursor is open.

### External procedures

A procedure using the **EXTERNAL NAME** clause is a wrapper around a call to an external dynamic link library, and is called an external stored procedure. An external stored procedure can have no clauses other than the **EXTERNAL NAME** clause following the parameter list. For a description of external procedures, see "Calling external libraries from stored procedures" in the chapter "Using Procedures, Triggers, and Batches".

### Example

- ◆ The following procedure uses a case statement to classify the results of a query.

```

CREATE PROCEDURE ProductType (IN product_id INT,
OUT type CHAR(10))
BEGIN
    DECLARE prod_name CHAR(20) ;
    SELECT name INTO prod_name FROM
    "DBA"."product"
    WHERE id = product_id;
    CASE prod_name
    WHEN 'Tee Shirt' THEN
        SET type = 'Shirt'
    WHEN 'Sweatshirt' THEN

```

## CREATE PROCEDURE statement

---

```
        SET type = 'Shirt'
    WHEN 'Baseball Cap' THEN
        SET type = 'Hat'
    WHEN 'Visor' THEN
        SET type = 'Hat'
    WHEN 'Shorts' THEN
        SET type = 'Shorts'
    ELSE
        SET type = 'UNKNOWN'
    END CASE ;
END
```

- ◆ The following procedure uses a cursor and loops over the rows of the cursor to return a single value.

```
CREATE PROCEDURE TopCustomer (OUT TopCompany
    CHAR(35), OUT TopValue INT)
BEGIN
    DECLARE err_notfound EXCEPTION
    FOR SQLSTATE '02000' ;
    DECLARE curThisCust CURSOR FOR
    SELECT company_name, CAST(
    sum(sales_order_items.quantity *
    product.unit_price) AS INTEGER) VALUE
    FROM customer
    LEFT OUTER JOIN sales_order
    LEFT OUTER JOIN sales_order_items
    LEFT OUTER JOIN product
    GROUP BY company_name ;

    DECLARE ThisValue INT ;
    DECLARE ThisCompany CHAR(35) ;
    SET TopValue = 0 ;
    OPEN curThisCust ;
    CustomerLoop:
    LOOP
        FETCH NEXT curThisCust
        INTO ThisCompany, ThisValue ;
        IF SQLSTATE = err_notfound THEN
            LEAVE CustomerLoop ;
        END IF ;
        IF ThisValue > TopValue THEN
            SET TopValue = ThisValue ;
            SET TopCompany = ThisCompany ;
        END IF ;
    END LOOP CustomerLoop ;
    CLOSE curThisCust ;
END
```



# CREATE PUBLICATION statement

<b>Function</b>	To create a publication for replication with SQL Remote.
<b>Syntax</b>	<b>CREATE PUBLICATION</b> [ <i>owner.</i> ] <i>publication-name</i> ... ( <b>TABLE</b> <i>article-description</i> ,... )
<b>Parameters</b>	<i>article-description</i> : <i>table-name</i> [ ( <i>column-name</i> , ... ) ] ... [ <b>WHERE</b> <i>search-condition</i> ] ... [ <b>SUBSCRIBE BY</b> <i>expression</i> ]
<b>Usage</b>	Anywhere. This statement is applicable only to SQL Remote.
<b>Permissions</b>	Must have DBA authority. Requires exclusive access to all tables referred to in the statement.
<b>Side effects</b>	Automatic commit.
<b>See also</b>	ALTER PUBLICATION statement DROP PUBLICATION statement
<b>Description</b>	<p>The CREATE PUBLICATION statement creates a SQL Remote publication in the database. A publication can be created for another user by specifying an owner name.</p> <p>In SQL Remote, publishing is a two-way operation, as data can be entered at both consolidated and remote databases. In a SQL Remote installation, any consolidated database and all remote databases must have the same publication defined. Running the extraction utility from a consolidated database automatically executes the correct CREATE PUBLICATION statement in the remote database.</p> <p><b>Article</b> Publications are built from articles. Each article is a table or part of a table. An article may be a vertical partition of a table (a subset of the table's columns), a horizontal partition (a subset of the table's rows) or a vertical and horizontal partition.</p> <p><b>SUBSCRIBE BY clause</b> One way of defining a subset of rows of a table to be included in an article is to use a SUBSCRIBE BY clause. This clause allows many different subscribers to receive different rows from a table in a single publication definition.</p> <p><b>WHERE clause</b> The WHERE clause is a way of defining the subset of rows of a table to be included in an article. It is useful if the same subset is to be received by all subscribers to the publication.</p>

You can combine WHERE and SUBSCRIBE BY clauses in an article definition.

**Example**

```
CREATE PUBLICATION pub_contact (  
  TABLE contact  
)
```

# CREATE REMOTE MESSAGE TYPE statement

<b>Function</b>	To identify a message-link and return address for outgoing messages from a database.
<b>Syntax</b>	<b>CREATE REMOTE MESSAGE TYPE</b> <i>message-system</i> ... <b>ADDRESS</b> <i>address-string</i>
<b>Parameters</b>	<i>message-system</i> : <b>MAP   FILE   VIM   SMTP</b>
<b>Usage</b>	Anywhere. This statement is applicable only to SQL Remote.
<b>Permissions</b>	Must have DBA authority.
<b>Side effects</b>	Automatic commit.
<b>See also</b>	GRANT PUBLISH statement GRANT REMOTE statement GRANT CONSOLIDATE statement "SQL Remote message types" in the chapter "SQL Remote Administration"
<b>Description</b>	<p>The Message Agent sends outgoing messages from a database using one of the supported message links. Return messages for users employing the specified link are sent to the specified address as long as the remote database is created by the extraction utility. The Message Agent starts links only if it has remote users for those links.</p> <p>The address is the publisher's address under the specified message system. If it is an e-mail system, the address string must be a valid e-mail address. If it is a file-sharing system, the address string is a subdirectory of the directory set in the SQLREMOTE environment variable, or of the current directory if that is not set. You can override this setting on the GRANT CONSOLIDATE statement at the remote database.</p> <p>For the FILE link, the CREATE REMOTE MESSAGE TYPE statement also causes the Message Agent to look for incoming messages in the address given for each message type.</p> <p>The initialization utility creates message types automatically, without an address. Unlike other CREATE statements, the CREATE REMOTE MESSAGE TYPE statement does not give an error if the type exists; instead it alters the type.</p>
<b>Example</b>	When remote databases are extracted using the extraction utility, the following statement sets all recipients of FILE messages to send messages back to the company subdirectory of the SQLREMOTE environment variable.

## *CREATE REMOTE MESSAGE TYPE statement*

---

The statement also instructs DBREMOTE to look in the COMPANY subdirectory for incoming messages.

```
CREATE REMOTE MESSAGE TYPE file  
ADDRESS 'company'
```

# CREATE SCHEMA statement

<b>Function</b>	Creates a collection of tables, views, and permissions for a database user.
<b>Syntax</b>	<pre> <b>CREATE SCHEMA AUTHORIZATION</b> <i>userid</i>     ...     [         <i>create-table-statement</i>           <i>create-view-statement</i>           <i>grant-statement</i>     ],... </pre>
<b>Usage</b>	Anywhere
<b>Transact-SQL Usage</b>	You cannot use the CREATE SCHEMA statement in a Transact-SQL batch or procedure.
<b>Permissions</b>	Must have RESOURCE authority.
<b>Side effects</b>	Automatic commit.
<b>See also</b>	CREATE TABLE statement CREATE VIEW statement GRANT statement
<b>Description</b>	<p>The CREATE SCHEMA statement creates a schema. A schema is a collection of tables, views, and their associated permissions.</p> <p>The <i>userid</i> must be the user ID of the current connection. You cannot create a schema for another user.</p> <p>If any of the statements contained in the CREATE SCHEMA statement fails, then the entire CREATE SCHEMA statement is rolled back.</p> <p>The CREATE SCHEMA statement is simply a way of collecting together individual CREATE and GRANT statements into one operation. There is no SCHEMA database object created in the database, and to drop the objects you must use individual DROP TABLE or DROP VIEW statements. To revoke permissions, you must use a REVOKE statement for each permission granted.</p> <p>The individual CREATE or GRANT statements are not separated by statement delimiters. The statement delimiter marks the end of the CREATE SCHEMA statement itself.</p> <p>The individual CREATE or GRANT statements must be ordered such that the objects are created before permissions are granted on them.</p>

Although the CREATE SCHEMA statement can currently be executed by a user more than once, this is not recommended, and may not be supported in future releases.

### **Examples**

The following CREATE SCHEMA statement creates a schema consisting of two tables. The statement must be executed by the user ID `sample_user`, who must have RESOURCE authority. If the statement creating table `t2` fails, neither table is created.

```
CREATE SCHEMA AUTHORIZATION sample_user
  CREATE TABLE t1 ( id1 INT PRIMARY KEY )
  CREATE TABLE t2 ( id2 INT PRIMARY KEY ) ;
```

The statement delimiter in the following CREATE SCHEMA statement is placed after the first CREATE TABLE statement. As the statement delimiter marks the end of the CREATE SCHEMA statement, the example is interpreted as a two statement batch by the database engine. Consequently, if the statement creating table `t2` fails, the table `t1` is still created.

```
CREATE SCHEMA AUTHORIZATION sample_user
  CREATE TABLE t1 ( id1 INT PRIMARY KEY ) ;
  CREATE TABLE t2 ( id2 INT PRIMARY KEY ) ;
```

# CREATE SUBSCRIPTION statement

<b>Function</b>	To create a subscription for a user to a publication.
<b>Syntax</b>	<pre>CREATE SUBSCRIPTION TO <i>publication-name</i> [ ( <i>string</i> ) ] ... FOR <i>userid</i></pre>
<b>Usage</b>	Anywhere. This statement is applicable only to SQL Remote.
<b>Permissions</b>	Must have DBA authority.
<b>Side effects</b>	Automatic commit.
<b>See also</b>	<p>DROP SUBSCRIPTION statement  GRANT REMOTE statement  SYNCHRONIZE SUBSCRIPTION statement  START SUBSCRIPTION statement  "Setting up subscriptions" in the chapter "SQL Remote Administration",</p>
<b>Description</b>	<p>In a SQL Remote installation, data is organized into <b>publications</b> for replication. In order to receive SQL Remote messages, a <b>subscription</b> must be created for a user ID with REMOTE permissions.</p> <p>If a string is supplied in the subscription, it is matched against each SUBSCRIBE BY expression in the publication. The subscriber receives all rows for which the value of the expression is equal to the supplied string.</p> <p>In SQL Remote, publications and subscriptions are two-way relationships. If you create a subscription for a remote user to a publication on a consolidated database, you should also create a subscription for the consolidated database on the remote database. The extraction utility carries this out automatically.</p>
<b>Example</b>	<pre>CREATE SUBSCRIPTION TO pub_sales ( 'Eastern' ) FOR p_chin</pre>

# CREATE TABLE statement

<b>Function</b>	To create a new table in the database.
<b>Syntax</b>	<b>CREATE</b> [ <b>GLOBAL TEMPORARY</b> ] <b>TABLE</b> [ <i>owner.</i> ] <i>table-name</i> ... ( { <i>column-definition</i> [ <i>column-constraint</i> ... ]   <i>table-constraint</i> }, ... ) ... [ { <b>IN</b>   <b>ON</b> } <i>dbspace-name</i> ] ... [ <b>ON COMMIT</b> { <b>DELETE</b>   <b>PRESERVE</b> } <b>ROWS</b> ]
<b>Parameters</b>	<i>column-definition</i> : <i>column-name</i> <i>data-type</i> [ <b>NOT NULL</b> ] [ <b>DEFAULT</b> <i>default-value</i> ]  <i>column-constraint</i> : <b>UNIQUE</b>   <b>PRIMARY KEY</b>   <b>REFERENCES</b> <i>table-name</i> [( <i>column-name</i> )] [ <i>actions</i> ]   <b>CHECK</b> ( <i>condition</i> )  <i>default-value</i> : <i>string</i>   <i>number</i>   <b>AUTOINCREMENT</b>   <b>CURRENT DATE</b>   <b>CURRENT TIME</b>   <b>CURRENT TIMESTAMP</b>   <b>NULL</b>   <b>USER</b>   ( <i>constant-expression</i> )  <i>table-constraint</i> : <b>UNIQUE</b> ( <i>column-name</i> , ... )   <b>PRIMARY KEY</b> ( <i>column-name</i> , ... )   <b>CHECK</b> ( <i>condition</i> )   <i>foreign-key-constraint</i>  <i>foreign-key-constraint</i> : [ <b>NOT NULL</b> ] <b>FOREIGN KEY</b> [ <i>role-name</i> ] [( <i>column-name</i> , ... )] ... <b>REFERENCES</b> <i>table-name</i> [( <i>column-name</i> , ... )] ... [ <i>actions</i> ] [ <b>CHECK ON COMMIT</b> ]  <i>action</i> : <b>ON</b> { <b>UPDATE</b>   <b>DELETE</b> } ...{ <b>CASCADE</b>   <b>SET NULL</b>   <b>SET DEFAULT</b>   <b>RESTRICT</b> }
<b>Usage</b>	Anywhere.
<b>Permissions</b>	Must have RESOURCE authority. To create a table for another user, you must have DBA authority.
<b>Side effects</b>	Automatic commit.



**See also**

DROP statement  
 ALTER TABLE statement  
 CREATE DBSPACE statement  
 "SQL Anywhere Data Types"  
 "Creating tables" in the chapter "Working with Database Objects"

**Description**

The CREATE TABLE statement creates a new table. A table can be created for another user by specifying an owner name. If GLOBAL TEMPORARY is not specified, the table is referred to as a **base table**. Otherwise, the table is a **temporary table**.

The IN clause is allowed only for base tables, and is used to specify in which database file the base table will be created. See "CREATE DBSPACE statement" on page 1004 for more information.

A created temporary table is a table that exists in the database like a base table and remains in the database until it is explicitly removed by a DROP TABLE statement. The rows in a temporary table are only visible to the connection that inserted the rows. Multiple connections from the same or different applications can use the same temporary table at the same time and each connection will only see its own rows. The rows of a temporary table are deleted when the connection ends.

The ON COMMIT clause is only allowed for temporary tables. By default, the rows of a temporary table are deleted on COMMIT.

The parenthesized list following the CREATE TABLE statement can contain the following clauses in any order:

**column-name data-type [ NOT NULL ] [ DEFAULT default-value ]**

Define a column in the table. Allowable data types are described in "SQL Anywhere Data Types". Two columns in the same table cannot have the same name.

If NOT NULL is specified, or if the column is in a UNIQUE or PRIMARY KEY constraint, the column cannot contain any NULL values. If a DEFAULT value is specified, it will be used as the value for the column in any INSERT statement which does not specify a value for the column. If no DEFAULT is specified, it is equivalent to DEFAULT NULL.

When using DEFAULT AUTOINCREMENT, the **data type** must be one of INTEGER, SMALLINT, FLOAT, or DOUBLE. On INSERTs into the table, if a value is not specified for the autoincrement column, a unique value is generated. If a value is specified, it will be used. If the value is larger than the current maximum value for the column, that value will be used as a starting point for subsequent INSERTs.

Deleting rows does not decrement the autoincrement counter. Gaps created by deleting rows can only be filled by explicit assignment when using an insert. After doing an explicit insert of a row number less than the maximum, subsequent rows without explicit assignment are autoincremented with a value of one greater than the previous maximum.

The next value to be used for each column is stored as a long integer (4 bytes). Using values greater than  $(2^{31} - 1)$ , that is, large double or numeric values, may cause wraparound to negative values, and AUTOINCREMENT should not be used in such cases.

For performance reasons, it is highly recommended that DEFAULT AUTOINCREMENT only be used with columns defined as a PRIMARY KEY or with a UNIQUE constraint; or columns that are the first column of an index. This will allow the maximum value determined at startup time to be found without scanning the entire table.

Constant expressions that do not reference database objects are allowed in a DEFAULT clause, so that functions such as getdate or dateadd can be used. If the expression is not a function or simple value, it must be enclosed in parentheses.

**table-constraint** Table constraints help ensure the integrity of data in the database. There are four types of integrity constraints:

- ◆ **Unique constraint** Identifies one or more columns that uniquely identify each row in the table.
- ◆ **Primary key constraint** is the same as a unique constraint except that a table can have only one primary key constraint. The primary key usually identifies the best identifier for a row. For example, the customer number might be the primary key for the customer table.
- ◆ **Foreign key constraint** restricts the values for a set of columns to match the values in a primary key or uniqueness constraint of another table. For example, a foreign key constraint could be used to ensure that a customer number in an invoice table corresponds to a customer number in the customer table.
- ◆ **Check constraint** allows arbitrary conditions to be verified. For example, a check constraint could be used to ensure that a column called Sex only contains the values male or female.

If a statement would cause changes to the database that would violate an integrity constraint, the statement is effectively not executed and an error is reported. (*Effectively* means that any changes made by the statement before the error was detected are undone.)

**column-constraint** Column constraints are abbreviations for the corresponding table constraints. For example, the following are equivalent:

```
CREATE TABLE Product (
    product_num integer UNIQUE
)

CREATE TABLE Product (
    product_num integer,
    UNIQUE ( product_num )
)
```

Column constraints are normally used unless the constraint references more than one column in the table. In these cases, a table constraint must be used.

## Integrity constraints

**column-definition UNIQUE or UNIQUE ( column-name, ... )** No two rows in the table can have the same values in all the named column(s). A table may have more than one unique constraint.

### Unique constraint versus unique index

There is a difference between a **unique constraint** and a **unique index**. Columns in a unique index are allowed to be NULL, while columns in a unique constraint are not. Also, the column referenced by a foreign key can be either a primary key or a column with a unique constraint. Unique indexes cannot be referenced, because they can include multiple NULLs.

**column-definition PRIMARY KEY or PRIMARY KEY ( column-name, ... )** The primary key for the table will consist of the listed column(s), and none of the named column(s) can contain any NULL values. SQL Anywhere ensures that each row in the table will have a unique primary key value. A table can have only one PRIMARY KEY.

When the second form is used (PRIMARY KEY followed by a list of columns), the primary key is created including the columns in the order in which they are defined, not the order in which they are listed.

**column-definition REFERENCES primary-table-name**

**[(primary-column-name)]** The column is a **foreign key** for the primary key or a unique constraint in the primary table. Normally, a foreign key would be for a primary key rather than a unique constraint. If a primary column name is specified, it must match a column in the primary table which is subject to a unique constraint or primary key constraint, and that constraint must consist of only that one column. Otherwise the foreign key references the primary key of the second table.

A temporary table cannot have a foreign key that references a base table and a base table cannot have a foreign key that references a temporary table.

**[ NOT NULL ] FOREIGN KEY [role-name] [(...)] REFERENCES primary-table-name [(...)]**

The table contains a **foreign key** for the primary key or a unique constraint in another table. Normally, a foreign key would be for a primary key rather than a unique constraint. (In this description, this other table will be called the **primary table**.)

If the primary table column names are not specified, then the primary table columns will be the columns in the table's primary key. If foreign key column names are not specified then the foreign key columns will have the same names as the columns in the primary table. If foreign key column names are specified, then the primary key column names must be specified, and the column names are paired according to position in the lists.

Any foreign key column not explicitly defined will automatically be created with the same data type as the corresponding column in the primary table. These automatically created columns cannot be part of the primary key of the foreign table. Thus, a column used in both a primary key and foreign key must be explicitly created.

A foreign key can be explicitly created to be NOT NULL. In this case, no row is allowed that is NULL for any column in the key. If NOT NULL is not explicitly stated, the foreign key is still automatically defined as NOT NULL when all the columns in the foreign key do not allow null values at the time the foreign key is created.

The **role name** is the name of the foreign key. The main function of the role name is to distinguish two foreign keys to the same table. If no role name is specified, the role name is assigned as follows:

- 1 If there is no foreign key with a role name the same as the table name, then the table name is assigned as the role name.
- 2 If the table name is already taken, the role name is the table name concatenated with a zero-padded three-digit number unique to the table.

The referential integrity action defines the action to be taken to maintain foreign key relationships in the database. Whenever a primary key value is changed or deleted from a database table, there may be corresponding foreign key values in other tables that should be modified in some way. You can specify either an ON UPDATE clause, an ON DELETE clause, or both, followed by one of the following actions:

**CASCADE** When used with ON UPDATE, update the corresponding foreign keys to match the new primary key value. When used with ON DELETE, deletes the rows from the table that match the deleted primary key.

**SET NULL** Sets to NULL all the foreign key values that correspond to the updated or deleted primary key.

**SET DEFAULT** Sets to the value specified by the column(s) DEFAULT clause, all the foreign key values that match the updated or deleted primary key value.

**RESTRICT** Generates an error if an attempt is made to update or delete a primary key value while there are corresponding foreign keys elsewhere in the database. This was the only form of referential integrity prior to Watcom SQL Version 4.0 and is the default action if no action is specified.

The CHECK ON COMMIT clause causes the database to wait for a COMMIT before checking the integrity of this foreign key, overriding the setting of the WAIT\_FOR\_COMMIT database option. CHECK ON COMMIT delays only the RESTRICT referential integrity action; it does not delay any other referential integrity action such as CASCADE or SET NULL. You can have a foreign key declared with CHECK ON COMMIT and a CASCADE or SET NULL referential action. In this case, inserts can be done in the foreign table before the corresponding row is inserted in the primary table as long as the row is inserted before the COMMIT.

If you use the short form of CHECK ON COMMIT then RESTRICT is implied.

A temporary table cannot have a foreign key that references a base table and a base table cannot have a foreign key that references a temporary table.

**column-definition CHECK ( condition ) or CHECK ( condition )** No row is allowed to fail the condition. If an INSERT or UPDATE statement would cause a row to fail the condition, the operation is not permitted and the effects of the statement are undone.

**When is the change rejected?**

The change is rejected only if the condition is FALSE; in particular, the change is allowed if the condition is UNKNOWN. (See "NULL value" on page 1110 and "Search conditions" in the chapter "Watcom-SQL Language Reference" for more information about TRUE, FALSE, and UNKNOWN conditions.)

**Examples**

The first two examples are for a library database.

Create a table for a library database to hold book information.

```
CREATE TABLE library_books (
  -- NOT NULL is assumed for primary key columns
  isbn CHAR(20)          PRIMARY KEY,
  copyright_date        DATE,
  title                 CHAR(100),
  author                CHAR(50),
  -- column(s) corresponding to primary key of room
  -- will be created
  FOREIGN KEY location REFERENCES room
)
```

Create a table for a library database to hold information on borrowed books.

```
CREATE TABLE borrowed_book (
  -- Default on insert is that book is borrowed today
  date_borrowed DATE NOT NULL DEFAULT CURRENT DATE,
  -- date_returned will be NULL until the book is
  -- returned
  date_returned    DATE,
  book             CHAR(20)
                 REFERENCES library_books (isbn),
  -- The check condition is UNKNOWN until
  -- the book is returned, which is allowed
  CHECK( date_returned >= date_borrowed )
)
```

The following example is for a sales database.

Create tables for a sales database to hold order and order item information.

```
CREATE TABLE Orders (
  order_num INTEGER NOT NULL PRIMARY KEY,
  date_ordered DATE,
  name CHAR(80)
) ;

CREATE TABLE Order_item (
  order_num    INTEGER NOT NULL,
  item_num     SMALLINT NOT NULL,
  PRIMARY KEY (order_num, item_num),
  -- When an order is deleted, delete all of its
  -- items.
  FOREIGN KEY (order_num)
```

```
REFERENCES Orders (order_num)
ON DELETE CASCADE
)
```

# CREATE TRIGGER statement

<b>Function</b>	To create a new trigger in the database.
<b>Syntax</b>	<pre>CREATE TRIGGER <i>trigger-name</i> <i>trigger-time</i> <i>trigger-event</i> [, <i>trigger-event</i>,...] ... [ ORDER <i>integer</i> ] ON <i>table-name</i> ... [ REFERENCING [ OLD AS <i>old-name</i> ]                 [ NEW AS <i>new-name</i> ] ]                 [ REMOTE AS <i>remote-name</i> ] ] ... [ FOR EACH { ROW   STATEMENT } ] ... [ WHEN ( <i>search-condition</i> ) ] ... [ IF UPDATE ( <i>column-name</i> ) THEN ... [ { AND   OR } UPDATE ( <i>column-name</i> ) ] ... ]         ... <i>compound-statement</i> ... [ ELSEIF UPDATE ( <i>column-name</i> ) THEN ... [ { AND   OR } UPDATE ( <i>column-name</i> ) ] ... ]         ... <i>compound-statement</i> ... END IF ] ]</pre>
<b>Parameters</b>	<p><i>trigger-time</i>: <b>BEFORE</b>   <b>AFTER</b>   <b>RESOLVE</b></p> <p><i>trigger-event</i>: <b>DELETE</b>   <b>INSERT</b>   <b>UPDATE</b>   <b>UPDATE OF</b> <i>column-list</i></p>
<b>Usage</b>	Anywhere.
<b>Permissions</b>	Must have RESOURCE authority and have ALTER permissions on the table, or must have DBA authority. CREATE TRIGGER puts a table lock on the table and thus requires exclusive use of the table.
<b>Side effects</b>	Automatic commit.
<b>See also</b>	Compound statements CREATE PROCEDURE statement DROP statement The chapter "Using Procedures, Triggers, and Batches"
<b>Description</b>	<p>The CREATE TRIGGER statement creates a trigger associated with a table in the database and stores the trigger in the database.</p> <p>Triggers can be triggered by one or more of the following events:</p> <ul style="list-style-type: none"><li>◆ <b>INSERT</b> Invoked whenever a new row is inserted into the table associated with the trigger.</li><li>◆ <b>DELETE</b> Invoked whenever a row of the associated table is deleted.</li><li>◆ <b>UPDATE</b> Invoked whenever a row of the associated table is updated.</li></ul>



- ◆ **UPDATE OF column-list** Invoked whenever a row of the associated table is updated and a column in the *column-list* has been modified.

**BEFORE UPDATE** triggers fire any time an update occurs on a row, regardless of whether or not the new value differs from the old value. **AFTER UPDATE** triggers will fire only if the new value is different from the old value.

Row and  
statement-level  
triggers

The trigger is declared as either a row-level trigger, in which case it executes before or after each row is modified, or as a statement-level trigger, in which case it executes after the entire triggering statement is completed.

Row-level triggers can be defined to execute **BEFORE** or **AFTER** the insert, update, or delete. Statement-level triggers execute **AFTER** the statement. The **RESOLVE** trigger time is for use with SQL Remote; it fires before row-level **UPDATE** or **UPDATE OF** column-lists only.

To declare a trigger as a row-level trigger, use the **FOR EACH ROW** clause. To declare a trigger as a statement-level trigger, you can either use a **FOR EACH STATEMENT** clause or omit the **FOR EACH** clause. For clarity, it is recommended that you enter the **FOR EACH STATEMENT** clause if declaring a statement-level trigger.

Order of firing

Triggers of the same type (insert, update, or delete) that fire at the same time (before, after, or resolve) can use the **ORDER** clause to determine the order that the triggers are fired.

Referencing  
deleted and  
inserted values

The **REFERENCING OLD** and **REFERENCING NEW** clauses allow you to refer to the deleted and inserted rows. For the purposes of this clause, an **UPDATE** is treated as a delete followed by an insert.

The **REFERENCING REMOTE** clause is for use with SQL Remote. It allows you to refer to the values in the **VERIFY** clause of an **UPDATE** statement. It should be used only with **RESOLVE UPDATE** or **RESOLVE UPDATE OF** column-list triggers.

The meaning of **REFERENCING OLD** and **REFERENCING NEW** differs, depending on whether the trigger is a row-level or a statement-level trigger. For row-level triggers, the **REFERENCING OLD** clause allows you to refer to the values in a row prior to an update or delete, and the **REFERENCING NEW** clause allows you to refer to the inserted or updated values. The **OLD** and **NEW** rows can be referenced in **BEFORE** and **AFTER** triggers. The **REFERENCING NEW** clause allows you to modify the new row in a **BEFORE** trigger before the insert or update operation takes place.

For statement-level triggers, the REFERENCING OLD and REFERENCING NEW clauses refer to declared temporary tables holding the old and new values of the rows. The default names for these tables are **deleted** and **inserted**.

The WHEN clause causes the trigger to fire only for rows where the search-condition evaluates to true.

Updating values  
with the same  
value

BEFORE UPDATE triggers fire any time an UPDATE occurs on a row, whether or not the new value differs from the old value. AFTER UPDATE triggers fire only if the new value is different from the old value.

**Example**

- ◆ When a new department head is appointed, update the **manager\_id** column for employees in that department.

```
CREATE TRIGGER
tr_manager BEFORE UPDATE OF dept_head_id ON
department
REFERENCING OLD AS old_dept
NEW AS new_dept
FOR EACH ROW
BEGIN
    UPDATE employee
    SET employee.manager_id=new_dept.dept_head_id
    WHERE employee.dept_id=old_dept.dept_id
END
```

# CREATE VARIABLE statement

<b>Function</b>	To create a SQL variable.
<b>Syntax</b>	<b>CREATE VARIABLE</b> <i>identifier data-type</i>
<b>Usage</b>	Anywhere.
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>See also</b>	Compound statements "SQL Anywhere Data Types" DROP VARIABLE statement SET statement
<b>Description</b>	<p>The CREATE VARIABLE statement creates a new variable of the specified data type. The variable contains the NULL value until it is assigned a different value by the SET VARIABLE statement.</p> <p>A variable can be used in a SQL statement anywhere a column name is allowed. If there is no column name that matches the identifier, SQL Anywhere checks to see if there is a variable that matches and uses its value.</p> <p>Variables belong to the current connection, and disappear when you disconnect from the database or when you use the DROP VARIABLE statement. Variables are not visible to other connections. Variables are not affected by COMMIT or ROLLBACK statements.</p> <p>Variables are useful for creating large text or binary objects for INSERT or UPDATE statements from Embedded SQL programs.</p> <p>Variables created by CREATE VARIABLE can be used in any SQL statement or in any procedure or trigger.</p> <p>Local variables in procedures and triggers are declared within a compound statement (see "Using compound statements" in the chapter "Using Procedures, Triggers, and Batches").</p>

## Example

- ◆ The following code fragment could be used to insert a large text value into the database.

```
EXEC SQL BEGIN DECLARE SECTION;
char buffer[5000];
EXEC SQL END DECLARE SECTION;
EXEC SQL CREATE VARIABLE hold_blob LONG VARCHAR;
EXEC SQL SET hold_blob = '';
for(;;) {
    /* read some data into buffer ... */
    size = fread( buffer, 1, 5000, fp );
```

## *CREATE VARIABLE statement*

---

```
    if( size <= 0 ) break;
    /* add data to blob using concatenation
    Note that concatenation works for binary
    data too! */
    EXEC SQL SET hold_blob = hold_blob || :buffer;
}
EXEC SQL INSERT INTO some_table VALUES ( 1,
hold_blob );
EXEC SQL DROP VARIABLE hold_blob;
```

# CREATE VIEW statement

<b>Function</b>	To create a view on the database. Views are used to give a different perspective on the data even though it is not stored that way.
<b>Syntax</b>	<pre> <b>CREATE VIEW</b> ... [ <i>owner.</i>view-name [( <i>column-name</i>, ... )] ... <b>AS</b> <i>select-without-order-by</i> ... [ <b>WITH CHECK OPTION</b> ] </pre>
<b>Usage</b>	Anywhere.
<b>Permissions</b>	Must have RESOURCE authority and SELECT permission on the tables in the view definition.
<b>Side effects</b>	Automatic commit.
<b>See also</b>	DROP statement CREATE TABLE statement
<b>Description</b>	<p>The CREATE VIEW statement creates a view with the given name. A view can be created for another user by specifying the <b>owner</b>. A view name can be used in place of a table name in SELECT, DELETE, UPDATE, and INSERT statements. Views, however, do not physically exist in the database as tables. They are derived each time they are used. The view is derived as the result of the SELECT statement specified in the CREATE VIEW statement. Table names used in a view should be qualified by the user ID of the table owner. Otherwise, a different user ID might not be able to find the table or might get the wrong table.</p> <p>The columns in the view are given the names specified in the column name list. If the column name list is not specified, then the view columns are given names from the select list items. In order to use the names from the select list items, the items must be a simple column name or they must have an alias-name specified (see "SELECT statement").</p> <p>Views can be updated provided the SELECT statement defining the view does not contain a GROUP BY clause, an aggregate function, or involve a UNION operation.</p> <p>The WITH CHECK OPTION clause rejects any updates and inserts to the view that do not meet the criteria of the view as defined by its SELECT statement.</p> <p>The SELECT statement must not have an ORDER BY clause on it. It may have a GROUP BY clause and may be a UNION.</p>

**Examples**

Create a view showing all information for male employees only. This view has the same column names as the base table.

```
CREATE VIEW male_employee
AS SELECT *
FROM Employee
WHERE Sex = 'M'
```

Create a view showing employees and the departments they belong to.

```
CREATE VIEW emp_dept
AS SELECT emp_lname, emp_fname, dept_name
FROM Employee JOIN Department
ON Employee.dept_id = Department.dept_id
```

# DBTOOL statement

## Function

To invoke one of the database tools.

## Syntax

**DBTOOL**  
*alter-database*  
 | *alter-writefile*  
 | *backup-to*  
 | *compress-database*  
 | *create-database*  
 | *create-writefile*  
 | *dbinfo-database*  
 | *drop-database*  
 | *translate*  
 | *uncompress-database*  
 | *unload-collation*  
 | *unload-tables*  
 | *validate-tables*

## Parameters

*alter-database:*

**ALTER DATABASE** *name*  
 ... { **NO** [ **TRANSACTION** ] **LOG**  
 | **SET** [ **TRANSACTION** ] **LOG TO** *filename* }

*alter-writefile:*

**ALTER WRITEFILE** *name* [ **REFER TO** *dbname* ]

*backup-to:*

**BACKUP TO** *directory*  
 ... { [ **DBFILE** ] [ **WRITE FILE** ] [ [ **TRANSACTION** ] **LOG** ]  
 | [ **ALL FILES** ] }  
 ... { [ **RENAME** [ **TRANSACTION** ] **LOG** ]  
 | [ **TRUNCATE** [ **TRANSACTION** ] **LOG** ] }  
 ... [ **NOCONFIRM** ] **USING** *connection-string*

*compress-database:*

**COMPRESS DATABASE** *filename* [ **TO** *filename* ]

*create-database:*

**CREATE DATABASE** *filename*  
 ... { [ **NO** [ **TRANSACTION** ] **LOG** ]  
 | [ [ **TRANSACTION** ] **LOG TO** *filename* ] }  
 ... [ { **IGNORE** | **RESPECT** } **CASE** ]  
 ... [ **PAGE SIZE** *n* ] [ **COLLATION** *name* ]  
 ... [ **ENCRYPT** ] [ **TRAILING SPACES** ]

*create-writefile:*

**CREATE WRITEFILE** *name* **FOR DATABASE** *name*  
 ... [ [ **TRANSACTION** ] **LOG TO** *logname* ] [ **NOCONFIRM** ]

*dbinfo-database:*

**DBINFO DATABASE** *filename*  
**TO** *filename* [ [ **WITH** ] **PAGE USAGE** ]

*drop-database:*

**DROP DATABASE** *name* [ **NOCONFIRM** ]

*translate:*

**TRANSLATE** [ **TRANSACTION** ] **LOG FROM** *logname*  
 ... [ **TO** *sqlfile* ] [ **WITH ROLLBACKS** ]  
 ... [ **USERS** *u1, u2, ... , un* ]  
 ... [ **EXCLUDE USERS** *u1, u2, ... , un* ]  
 ... [ **LAST CHECKPOINT** ] [ **ANSI** ] [ **NOCONFIRM** ]

*uncompress-database:*

**UNCOMPRESS DATABASE** *filename* [ **TO** *filename* ] [ **NOCONFIRM** ]

*unload-collation:*

**UNLOAD COLLATION** [ *name* ] **TO** *filename*  
 ... **USING** *connection-string*  
 ... [ **EMPTY MAPPINGS** ] [ **HEX** | **HEXADECIMAL** ] [ **NOCONFIRM** ]

*unload-tables:*

**UNLOAD TABLES TO** *directory* [ **RELOAD FILE TO** *filename* ]  
 ... [ **DATA** | **SCHEMA** ]  
 ... [ **UNORDERED** ] [ **VERBOSE** ] **USING** *connection-string*

*validate-tables:*

**VALIDATE TABLES** [ *t1, t2, ..., tn* ] **USING** *connection-string*

*connection-string:*

*string of connection parameters*

**Usage**

ISQL.

**Permissions**

None.

**Side effects**

None.

**See also**

The chapter "SQL Anywhere Components"

**Description**

The DBTOOL statement invokes one of the database utilities. All of the database utilities are available without leaving ISQL.

The following table lists the database utility invoked by each DBTOOL statement.

☞ For more information on the database utility programs, see the chapter "SQL Anywhere Components".

Statement	Database tool
DBTOOL ALTER DATABASE	DBLOG



---

<b>Statement</b>	<b>Database tool</b>
DBTOOL ALTER WRITEFILE	DBWRITE
DBTOOL BACKUP TO	DBBACKUP
DBTOOL COMPRESS DATABASE	DBSHRINK
DBTOOL CREATE DATABASE	DBINIT
DBTOOL CREATE WRITEFILE	DBWRITE
DBTOOL DBINFO DATABASE	DBINFO
DBTOOL DROP DATABASE	DBERASE
DBTOOL TRANSLATE	DBTRAN
DBTOOL UNCOMPRESS DATABASE	DBEXPAND
DBTOOL UNLOAD COLLATION	DBCOLLAT
DBTOOL UNLOAD TABLES	DBUNLOAD
DBTOOL VALIDATE TABLES	DBVALID

## Declaration section

<b>Function</b>	To declare host variables in an Embedded SQL program. Host variables are used to exchange data with the database.
<b>Syntax</b>	<b>EXEC SQL BEGIN DECLARE SECTION;</b> C declarations <b>EXEC SQL END DECLARE SECTION;</b>
<b>Usage</b>	Embedded SQL.
<b>Permissions</b>	None.
<b>See also</b>	"Host variables" in the chapter "The Embedded SQL Interface"
<b>Description</b>	A declaration section is simply a section of C variable declarations surrounded by the <b>BEGIN DECLARE SECTION</b> and <b>END DECLARE SECTION</b> statements. A declaration section makes the SQL preprocessor aware of C variables that will be used as host variables. Not all C declarations are valid inside a declaration section. See "Host variables" in the chapter "The Embedded SQL Interface" for more information.
<b>Examples</b>	<pre>EXEC SQL BEGIN DECLARE SECTION; char *emp_lname, initials[5]; int dept; EXEC SQL END DECLARE SECTION;</pre>

# DECLARE CURSOR statement

<b>Function</b>	To declare a cursor. Cursors are the primary means for retrieving data from the database using Embedded SQL.
<b>Syntax</b>	<pre> <b>DECLARE</b> <i>cursor-name</i> ... [ <b>UNIQUE</b>         <b>SCROLL</b>         <b>NO SCROLL</b>         <b>DYNAMIC SCROLL</b>         <b>INSENSITIVE</b> ] ... <b>CURSOR FOR</b> <i>statement</i>         <b>CURSOR FOR</b> <i>statement-name</i> ... [ <b>FOR UPDATE</b>   <b>FOR READ ONLY</b> ] </pre>
<b>Parameters</b>	<p><i>cursor-name</i>: <i>identifier</i></p> <p><i>statement-name</i>: <i>identifier</i>, or <i>host-variable</i></p>
<b>Usage</b>	<p>Embedded SQL, procedures, triggers, and batches.</p> <p>The <b>statement-name</b> and <b>host-variable</b> formats are for Embedded SQL only.</p>
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>See also</b>	<p>PREPARE statement</p> <p>OPEN statement</p> <p>EXPLAIN statement</p> <p>SELECT statement</p> <p>CALL statement</p>
<b>Description</b>	<p>The DECLARE CURSOR statement declares a cursor with the specified name for a SELECT statement or a CALL statement.</p> <p>When a cursor is declared UNIQUE, the query is forced to return all the columns required to uniquely identify each row. Often this will mean ensuring that all of the columns in the primary key or a uniqueness table constraint are returned. Any columns that are required but were not specified will be added. A DESCRIBE done on a UNIQUE cursor sets the following additional flags in the indicator variables:</p> <ul style="list-style-type: none"> <li>◆ DT_KEY_COLUMN The column is part of the key for the row</li> <li>◆ DT_HIDDEN_COLUMN The column was added to the query, since it was required to uniquely identify the rows</li> </ul>

A cursor declared FOR READ ONLY may not be used in an UPDATE (positioned) or a DELETE (positioned) operation. FOR UPDATE is the default.

A cursor declared NO SCROLL is restricted to FETCH NEXT and FETCH RELATIVE 0 seek operations. A cursor declared SCROLL or DYNAMIC SCROLL can use all formats of the FETCH statement. DYNAMIC SCROLL is the default.

SCROLL cursors behave differently from DYNAMIC SCROLL cursors when the rows in the cursor are modified or deleted after the first time the row is read. SCROLL cursors have more predictable behavior when changes happen.

Each row fetched in a SCROLL cursor is remembered. If one of these rows is deleted, either by your program or by another program in a multiuser environment, it creates a "hole" in the cursor. If you fetch the row at this "hole" with a SCROLL cursor, SQL Anywhere returns the error `SQL_NO_CURRENT_ROW` indicating that the row has been deleted, and leaves the cursor positioned on the "hole". (A DYNAMIC SCROLL cursor will just skip the "hole" and retrieve the next row.) This allows your application to remember row positions within a cursor and be assured that these positions will not change. For example, an application could remember that Cobb is the second row in the cursor for *SELECT \* FROM employee*. If the first employee (Whitney) is deleted while the SCROLL cursor is still open, `FETCH ABSOLUTE 2` will still position on Cobb while `FETCH ABSOLUTE 1` will return `SQL_NO_CURRENT_ROW`. Similarly, if the cursor is on Cobb, `FETCH PREVIOUS` will return `SQL_NO_CURRENT_ROW`.

In addition, a fetch on a SCROLL cursor will return the warning `SQL_ROW_UPDATED_WARNING` if the row has changed since it was last read. (The warning only happens once; fetching the same row a third time will not produce the warning.) Similarly, an UPDATE (positioned) or DELETE (positioned) statement on a row that has been modified since it was last fetched will return the error `SQL_ROW_UPDATED_SINCE_READ` and abort the statement. An application must FETCH the row again before the UPDATE or DELETE will be permitted. Note that an update to any column will cause the warning/error, even if the column is not referenced by the cursor. For example, a cursor on Surname and Initials would report the update even if only the Birthdate column were modified. These update warning and error conditions will not occur in bulk operations mode (-b database engine statement line switch) when row locking is disabled. See "Tuning bulk operations" in the chapter "Importing and Exporting Data".

SQL Anywhere maintains more information about SCROLL cursors than DYNAMIC SCROLL cursors; thus, DYNAMIC SCROLL cursors are more efficient and should be used unless the consistent behavior of SCROLL cursors is required. There is no extra overhead in SQL Anywhere for DYNAMIC SCROLL cursors versus NO SCROLL cursors.

#### **Scroll cursors**

The behavior of SCROLL cursors has changed since Watcom SQL 3.0. The SCROLL cursors in Watcom SQL 3.0 were equivalent to DYNAMIC SCROLL cursors now. Cursors declared with the default scrolling behavior will not have changed since the old default behavior was the SCROLL and the new default behavior is DYNAMIC SCROLL.

A cursor declared INSENSITIVE has its membership fixed when it is opened; a temporary table is created with a copy of all the original rows. FETCHING from an INSENSITIVE cursor does not see the effect of any other INSERT, UPDATE, or DELETE statement, or any other PUT, UPDATE WHERE CURRENT, DELETE WHERE CURRENT operations on a different cursor. It does see the effect of PUT, UPDATE WHERE CURRENT, DELETE WHERE CURRENT operations on the same cursor.

INSENSITIVE cursors make it easier to write an application that deals with cursors, since you only have to worry about changes you make explicitly to the cursor; you do not have to worry about actions taken by other users or by other parts of your application.

INSENSITIVE cursors can be expensive if the cursor is on a lot of rows. Also, INSENSITIVE cursors are not affected by ROLLBACK or ROLLBACK TO SAVEPOINT; the ROLLBACK is not an operation on the cursor that changes the cursor contents.

INSENSITIVE cursors meet the ODBC requirements for static cursors.

#### **Embedded SQL**

Statements are named using the PREPARE statement. Cursors can be declared only for a prepared SELECT or CALL.

The DECLARE cursor statement does not generate any C code.

**Cursor-name** is a string and is supplied by the programmer.

#### **Embedded SQL Examples**

```
1. EXEC SQL DECLARE cur_employee SCROLL CURSOR FOR
   SELECT * FROM employee ;

2. EXEC SQL PREPARE employee_statement
   FROM 'SELECT emp_lname FROM employee' ;
   EXEC SQL DECLARE cur_employee CURSOR FOR
   employee_statement ;
```

## *DECLARE CURSOR statement*

---

### **Procedure/trigger Example**

```
BEGIN
DECLARE cur_employee CURSOR FOR
    SELECT emp_lname
        FROM employee ;
DECLARE name CHAR(40) ;
OPEN cur_employee;
LOOP
FETCH NEXT cur_employee into name ;
    ...
END LOOP
CLOSE cur_employee;
END
```

# DECLARE LOCAL TEMPORARY TABLE statement

<b>Function</b>	To declare a local temporary table.
<b>Syntax</b>	<pre><b>DECLARE LOCAL TEMPORARY TABLE</b> <i>table-name</i> ... ( { <i>column-definition</i> [ <i>column-constraint</i> ... ]   <i>table-constraint</i> }, ... ) ... [   <b>ON COMMIT DELETE ROWS</b>   ]   <b>ON COMMIT PRESERVE ROWS</b>  </pre>
<b>Usage</b>	Embedded SQL, procedures, triggers, and batches.
<b>Permissions</b>	Must have RESOURCE authority.
<b>Side effects</b>	None.
<b>See also</b>	CREATE TABLE statement "Using compound statements" in the chapter "Using Procedures, Triggers, and Batches"
<b>Description</b>	<p>The DECLARE LOCAL TEMPORARY TABLE statement declares a temporary table. See "CREATE TABLE statement" on page 1020 for definitions of <b>column-definition</b>, <b>column-constraint</b>, and <b>table-constraint</b>.</p> <p>Declared local temporary tables within compound statements exist within the compound statement. (See "Using compound statements" in the chapter "Using Procedures, Triggers, and Batches"). Otherwise, the declared local temporary table exists until the end of the connection.</p> <p>By default, the rows of a temporary table are deleted on COMMIT.</p>
<b>Embedded SQL Example</b>	<pre>1. EXEC SQL DECLARE LOCAL TEMPORARY TABLE MyTable (       number INT     );</pre>
<b>Procedure/trigger Example</b>	<pre>BEGIN   DECLARE LOCAL TEMPORARY TABLE TempTab (     number INT   );   ... END</pre>

## **DEALLOCATE DESCRIPTOR statement**

<b>Function</b>	Frees memory associated with a SQL descriptor area.
<b>Syntax</b>	<b>DEALLOCATE DESCRIPTOR</b> <i>descriptor-name</i> ...
<b>Usage</b>	Embedded SQL.
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>See also</b>	ALLOCATE DESCRIPTOR statement "The SQL descriptor area (SQLDA)" in the chapter "The Embedded SQL Interface"
<b>Description</b>	Frees all memory associated with a descriptor area, including the data items, indicator variables, and the structure itself.
<b>Example</b>	For an example, see "ALLOCATE DESCRIPTOR statement" on page 970.



# DELETE statement

<b>Function</b>	To delete rows from the database.
<b>Syntax</b>	<pre>DELETE [FROM] [ owner,]table-name ... [FROM table-list] ... [WHERE search-condition]</pre>
<b>Usage</b>	Anywhere.
<b>Permissions</b>	Must have DELETE permission on the table.
<b>Side effects</b>	None.
<b>See also</b>	TRUNCATE TABLE statement INSERT statement INPUT statement FROM clause
<b>Description</b>	<p>The DELETE statement deletes all the rows from the named table that satisfy the search condition. If no WHERE clause is specified, all rows from the named table are deleted.</p> <p>The DELETE statement can be used on views provided the SELECT statement defining the view has only one table in the FROM clause and does not contain a GROUP BY clause, an aggregate function, or involve a UNION operation.</p> <p>☞ For a full description of the FROM clause and joins, see "FROM clause" on page 1074.</p> <p>The optional second FROM clause in the DELETE statement allows rows to be deleted based on joins. If the second FROM clause is present, the WHERE clause qualifies the rows of this second FROM clause. Rows are deleted from the table name given in the first FROM clause.</p>
<b>Correlation name resolution</b>	<p>The following statement illustrates a potential ambiguity in table names in DELETE statements with two FROM clauses that use correlation names:</p> <pre>DELETE FROM table_1 FROM table_1 AS alias_1, table_2 AS alias_2 WHERE ...</pre> <p>The table <b>table_1</b> is identified without a correlation name in the first FROM clause, but with a correlation name in the second FROM clause. In this case, <b>table_1</b> in the first clause is identified with <b>alias_1</b> in the second clause—there is only one instance of <b>table_1</b> in this statement.</p>

This is an exception to the general rule that where a table is identified with a correlation name and without a correlation name in the same statement, two instances of the table are considered.

Consider the following example:

```
DELETE
FROM table_1
FROM table_1 AS alias_1, table_1 AS alias_2
WHERE ...
```

In this case, there are two instances of **table\_1** in the second from clause. In this case, there is no way of identifying which instance the first FROM clause should be identified with. The usual rules of correlation names apply, and **table\_1** in the first FROM clause is identified with neither instance in the second clause: there are three instances of **table\_1** in the statement.

### Examples

Remove employee 105 from the database.

```
DELETE
FROM employee
WHERE emp_id = 105
```

Remove all data prior to 1993 from the fin\_data table.

```
DELETE
FROM fin_data
WHERE year < 1993
```

Remove all names from the contact table if they are already present in the customer table.

```
DELETE
FROM contact
FROM contact, customer
WHERE contact.last_name = customer.lname
AND contact.first_name = customer.fname
```

## DELETE (positioned) statement

<b>Function</b>	To delete the data at the current location of a cursor.
<b>Syntax</b>	<b>DELETE</b> [ <b>FROM</b> <i>table-spec</i> ] ... <b>WHERE CURRENT OF</b> <i>cursor-name</i>
<b>Parameters</b>	<i>cursor-name</i> : <i>identifier</i> , or <i>host-variable</i> <i>table-spec</i> : [ <i>owner</i> .] <i>correlation-name</i> ] <i>owner</i> : <i>identifier</i>
<b>Usage</b>	Embedded SQL, procedures, triggers, and batches. The <b>host-variable</b> format is for Embedded SQL only.
<b>Permissions</b>	Must have DELETE permission on tables used in the cursor.
<b>Side effects</b>	None.
<b>See also</b>	UPDATE statement UPDATE (positioned) statement INSERT statement PUT statement
<b>Description</b>	This form of the DELETE statement deletes the current row of the specified cursor. The current row is defined to be the last row fetched from the cursor.
<b>Name resolution</b>	The table from which rows are deleted is determined as follows: <ul style="list-style-type: none"> <li>◆ If no FROM clause is included, the cursor must be on a single table only.</li> <li>◆ If the cursor is for a joined query (including using a view containing a join), then the FROM clause must be used. Only the current row of the specified table is deleted. The other tables involved in the join are not affected.</li> <li>◆ If a FROM clause is included, and no table owner is specified, the table-spec value is first matched against any correlation names. <ul style="list-style-type: none"> <li>◆ If a correlation name exists, the table-spec value is identified with the correlation name.</li> <li>◆ If a correlation name does not exist, the table-spec value must be unambiguously identifiable as a table name in the cursor.</li> </ul> </li> <li>◆ If a FROM clause is included, and a table owner is specified, the table-spec value must be unambiguously identifiable as a table name in the cursor.</li> </ul>

## *DELETE (positioned) statement*

---

- ◆ The positioned DELETE statement can be used on a cursor open on a view as long as the view is updatable.

### **Example**

- ◆ The following statement removes the current row from the database.

```
DELETE  
WHERE CURRENT OF cur_employee
```

# DESCRIBE statement

<b>Function</b>	To get information about the host variables required to store data retrieved from the database or host variables used to pass data to the database.
<b>Syntax</b>	<pre> DESCRIBE ... [ ALL         BIND VARIABLES FOR         INPUT         OUTPUT         SELECT LIST FOR       ] ... [ LONG NAMES [<i>long-name-spec</i>]   WITH VARIABLE RESULT ] ... [ FOR ] { <i>statement-name</i>   CURSOR <i>cursor-name</i> } ... INTO <i>sqlda-name</i> </pre>
<b>Parameters</b>	<p><i>long-name-spec</i>:  <b>OWNER.TABLE.COLUMN   TABLE.COLUMN   COLUMN</b></p> <p><i>statement-name</i>: <i>identifier</i>, or <i>host-variable</i></p> <p><i>cursor-name</i>: <i>declared cursor</i></p> <p><i>sqlda-name</i>: <i>identifier</i></p>
<b>Usage</b>	Embedded SQL.
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>See also</b>	DECLARE CURSOR statement PREPARE statement
<b>Description</b>	<p>The DESCRIBE statement sets up the named SQLDA to describe either the OUTPUT (equivalently SELECT LIST) or the INPUT (BIND VARIABLES) for the named statement. In the INPUT case, DESCRIBE BIND VARIABLES does not set up the data types in the SQLDA: this needs to be done by the application. The ALL keyword allows you to describe INPUT and OUTPUT in one SQLDA. If describing a statement, the statement must have been previously prepared using the prepare statement with the same statement name and the SQLDA must have been previously allocated (see function <b>alloc_sqlda</b>).</p> <p>If describing a cursor, the cursor must have been previously declared. The default action is to describe the OUTPUT. Only SELECT statements and CALL statements have OUTPUT. A DESCRIBE OUTPUT on any other statement will indicate no output by setting the <b>sqlid</b> field of the SQLDA to zero.</p>

**SELECT** The DESCRIBE OUTPUT statement fills in the data type and length in the SQLDA for each select list item. The name field is also filled in with a name for the select list item. If an alias is specified for a select list item, the name will be that alias. Otherwise, the name will be derived from the select list item: if the item is a simple column name, it will be used, otherwise, a substring of the expression will be used. DESCRIBE will also put the number of select list items in the **sqld** field of the SQLDA.

When the statement being described is a UNION of two or more SELECT statements, the column names returned for DESCRIBE OUTPUT are the same column names which would be returned for the first SELECT statement.

**CALL** The DESCRIBE OUTPUT statement fills in the data type, length, and name in the SQLDA for each INOUT or OUT parameter in the procedure. DESCRIBE OUTPUT will also put the number of INOUT or OUT parameters in the **sqld** field of the SQLDA.

**CALL (result set)** The DESCRIBE OUTPUT statement fills in the data type, length, and name in the SQLDA for each RESULT column in the procedure definition. DESCRIBE OUTPUT will also put the number of result columns in the **sqld** field of the SQLDA.

A **bind variable** is a value supplied by the application when the database executes the statements. Bind variables can be considered parameters to the statement. DESCRIBE INPUT will fill in the name fields in the SQLDA with the bind variable names. DESCRIBE INPUT will also put the number of bind variables in the **sqld** field of the SQLDA.

DESCRIBE uses the indicator variables in the SQLDA to provide additional information. DT\_PROCEDURE\_IN and DT\_PROCEDURE\_OUT are bits that are set in the indicator variable when a CALL statement is described. DT\_PROCEDURE\_IN indicates an IN or INOUT parameter and DT\_PROCEDURE\_OUT indicates an INOUT or OUT parameter. Procedure RESULT columns will have both bits clear. After a describe OUTPUT, these bits can be used to distinguish between statements that have result sets (need to use OPEN, FETCH, RESUME, CLOSE) and statements that do not (need to use EXECUTE). DESCRIBE INPUT only sets DT\_PROCEDURE\_IN and DT\_PROCEDURE\_OUT appropriately when a bind variable is an argument to a CALL statement; bind variables within an expression that is an argument in a CALL statement will not set the bits.

DESCRIBE ALL allows you to describe INPUT and OUTPUT with one request to the database engine. This has a performance benefit in a multi-user environment. The INPUT information will be filled in the SQLDA first, followed by the OUTPUT information. The **sqld** field contains the total number of INPUT and OUTPUT variables. The DT\_DESCRIBE\_INPUT bit in the indicator variable is set for INPUT variables and clear for OUTPUT variables.

### Retrieving long column names

The LONG NAMES clause is provided to retrieve column names for a statement or cursor. Without this clause, there is a 29-character limit on the length of column names: with the clause, names of an arbitrary length are supported.

If LONG NAMES is used, the long names are placed into the SQLDATA field of the SQLDA, as if you were fetching from a cursor. None of the other fields (SQLLEN, SQLTYPE, and so on) are filled in. The SQLDA must be set up like a FETCH SQLDA: it must contain one entry for each column, and the entry must be a string type.

The default specification for the long names is TABLE.COLUMN.

### Describing variable result sets

The WITH VARIABLE RESULT statement is used to describe procedures that may have more than one result set, with different numbers or types of columns.

If WITH VARIABLE RESULT is used, the database engine sets the SQLCOUNT value after the describe to one of the following values:

- ◆ **0** The result set may change: the procedure call should be described again following each OPEN statement.
- ◆ **1** The result set is fixed. No redescrining is required.

☞ For more information on the use of the SQLDA structure, see "The SQL descriptor area (SQLDA)" in the chapter "The Embedded SQL Interface".

### Example

```
sqllda = alloc_sqllda( 3 );
EXEC SQL DESCRIBE OUTPUT FOR employee_statement INTO
sqllda;
if( sqllda->.sqld > sqllda->.sqln ) {
actual_size = sqllda->.sqld;
free_sqllda( sqllda );
sqllda = alloc_sqllda( actual_size );
EXEC SQL DESCRIBE OUTPUT FOR employee_statement INTO
sqllda;
}
```

# DISCONNECT statement

**Function** To drop a connection with the database.

**Syntax** **DISCONNECT**  
*connection-name*  
| [ **CURRENT** ]  
| **ALL**

**Parameters** *connection-name*: *identifier, string or host-variable.*

**Usage** ISQL.

**Permissions** None.

**Side effects** None.

**See also** CONNECT statement  
SET CONNECTION statement

**Description** The DISCONNECT statement drops a connection with the database engine and releases all resources used by it. If the connection to be dropped was named on the CONNECT statement, then the name can be specified. Specifying ALL will drop all of the application's connections to all database environments. CURRENT is the default and will drop the current connection.

An implicit ROLLBACK is executed on connections that are dropped.

**Embedded SQL Example** EXEC SQL DISCONNECT :conn\_name  
**ISQL Example** DISCONNECT ALL



# DROP statement

<b>Function</b>	To remove objects from the database.
<b>Syntax</b>	<pre> <b>DROP</b>   { <b>DATATYPE</b>   <b>DOMAIN</b> } <i>datatype-name</i>     <b>DBSPACE</b> <i>dbspace-name</i>     <b>INDEX</b> [[<i>owner</i>].<i>table-name</i>.]<i>index-name</i>     <b>TABLE</b> [ <i>owner</i>.]<i>table-name</i>     <b>VIEW</b> [ <i>owner</i>.]<i>view-name</i>     <b>PROCEDURE</b> [ <i>owner</i>.]<i>procedure-name</i>     <b>FUNCTION</b> [ <i>owner</i>.]<i>function-name</i>     <b>TRIGGER</b> [[ <i>owner</i>.]<i>table-name</i>.]<i>trigger-name</i> </pre>
<b>Usage</b>	Anywhere.
<b>Permissions</b>	<p>For DROP DBSPACE, must have DBA authority and must be the only connection to the database.</p> <p>For DROP TRIGGER, must have ALTER permissions on the table.</p> <p>For others, must be the owner of the object, or have DBA authority.</p>
<b>Side effects</b>	Automatic commit. Clears the Data window in ISQL. DROP TABLE and DROP INDEX close all cursors for the current connection.
<b>See also</b>	<p>CREATE DATATYPE statement</p> <p>CREATE DBSPACE statement</p> <p>CREATE TABLE statement</p> <p>CREATE INDEX statement</p> <p>CREATE VIEW statement</p> <p>CREATE PROCEDURE statement</p> <p>CREATE FUNCTION statement</p> <p>CREATE TRIGGER statement</p> <p>ALTER TABLE statement</p>
<b>Description</b>	<p>The DROP statement removes the definition of the indicated database structure. If the structure is a dbspace, then all tables in that dbspace must be dropped prior to dropping the dbspace. If the structure is a table, all data in the table is automatically deleted as part of the dropping process. Also, all indexes and keys for the table are dropped by the DROP TABLE statement.</p> <p>DROP TABLE, DROP INDEX and DROP DBSPACE are prevented whenever the statement affects a table that is currently being used by another connection.</p> <p>DROP PROCEDURE and DROP FUNCTION are prevented when the procedure is in use by another connection.</p>

DROP DATATYPE is prevented if the data type is used in a table. You must change data types on all columns defined on the user-defined data type in order to drop the data type. It is recommended that you use DROP DOMAIN rather than DROP DATATYPE, as DROP DOMAIN is the syntax used in the ANSI/ISO SQL3 draft.

**Examples**

Drop the department table from the database.

```
DROP TABLE department
```

Drop the emp\_dept view from the database.

```
DROP VIEW emp_dept
```

## DROP CONNECTION statement

<b>Function</b>	To drop a connection to the database, belonging to any user.
<b>Syntax</b>	<b>DROP CONNECTION</b> <i>connection-id</i>
<b>Usage</b>	Anywhere. You must be connected to the same database on the same server as the connection ID to execute this statement.
<b>Permissions</b>	Must have DBA authority.
<b>Side effects</b>	None.
<b>See also</b>	CONNECT statement
<b>Description</b>	<p>The DROP CONNECTION statement disconnects a user from the database by dropping the connection to the database.</p> <p>The <i>connection-id</i> for the connection is obtained using the <b>connection_property</b> function to request the connection number. The following statement returns the connection ID of the current connection:</p> <pre>SELECT connection_property( 'number' )</pre>
<b>Example</b>	<p>The following statement drops connection with ID number 4.</p> <pre>DROP CONNECTION 4</pre>

## **DROP OPTIMIZER STATISTICS statement**

<b>Function</b>	To reset optimizer statistics.
<b>Syntax</b>	<b>DROP OPTIMIZER STATISTICS</b>
<b>Usage</b>	Anywhere.
<b>Permissions</b>	Must have DBA authority.
<b>Side effects</b>	None.
<b>Description</b>	The DROP OPTIMIZER STATISTICS statement resets optimizer statistics. The SQL Anywhere optimizer maintains statistics as it evaluates queries and uses these statistics to make better decisions optimizing subsequent queries. These statistics are stored permanently in the database. The DROP OPTIMIZER STATISTICS statement resets these statistics to default values. This statement is most useful when first learning about the optimizer. It helps to better understand the process used by the optimizer.

## **DROP PUBLICATION statement**

<b>Function</b>	To drop a SQL Remote publication.
<b>Syntax</b>	<b>DROP PUBLICATION</b> [ <i>owner.</i> ] <i>publication-name</i>
<b>Usage</b>	Anywhere. This statement is applicable only to SQL Remote.
<b>Permissions</b>	Must have DBA authority.
<b>Side effects</b>	Automatic commit. All subscriptions to the publication are dropped.
<b>See also</b>	CREATE PUBLICATION statement
<b>Description</b>	<p>The DROP PUBLICATION statement drops an existing publication from the database.</p> <p>Publication definitions are held at both consolidated and remote databases in a SQL Remote installation.</p>
<b>Example</b>	<pre>DROP PUBLICATION pub_contact</pre>

## **DROP REMOTE MESSAGE TYPE statement**

<b>Function</b>	To delete a message type definition from a database.
<b>Syntax</b>	<b>DROP REMOTE MESSAGE TYPE</b> <i>message-system</i>
<b>Parameters</b>	<i>message-system</i> : <b>MAP   FILE   VIM   SMTP</b>
<b>Usage</b>	Anywhere. This statement is applicable only to SQL Remote.
<b>Permissions</b>	Must have DBA authority. To be able to drop the type, there must be no user granted REMOTE or CONSOLIDATE permissions with this type.
<b>Side effects</b>	Automatic commit.
<b>See also</b>	CREATE REMOTE MESSAGE TYPE statement ALTER REMOTE MESSAGE TYPE statement "SQL Remote message types" in the chapter "SQL Remote Administration",
<b>Description</b>	The statement removes a message type from a database.
<b>Example</b>	The following statement drops the FILE message type from a database. <pre>DROP REMOTE MESSAGE TYPE file</pre>

# DROP STATEMENT statement

<b>Function</b>	To free statement resources.
<b>Syntax</b>	<b>DROP STATEMENT</b> [ <i>owner.</i> ] <i>statement-name</i>
<b>Parameters</b>	<i>statement-name</i> : <i>identifier</i> , or <i>host-variable</i>
<b>Usage</b>	Embedded SQL.
<b>Permissions</b>	Must have prepared the statement.
<b>Side effects</b>	None.
<b>See also</b>	PREPARE statement
<b>Description</b>	The DROP STATEMENT statement frees resources used by the named prepared statement. These resources are allocated by a successful PREPARE statement, and are normally not freed until the database connection is released.
<b>Example</b>	<pre>1. EXEC SQL DROP STATEMENT S1; 2. EXEC SQL DROP STATEMENT :stmt;</pre>

## **DROP VARIABLE statement**

<b>Function</b>	To eliminate a SQL variable.
<b>Syntax</b>	<b>DROP VARIABLE</b> <i>identifier</i>
<b>Usage</b>	Anywhere.
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>See also</b>	CREATE VARIABLE statement SET statement
<b>Description</b>	The DROP VARIABLE statement eliminates a SQL variable previously created using the CREATE VARIABLE statement. Variables will be automatically eliminated when the database connection is released. However, variables are often used for large objects. Thus, eliminating them after use may free up significant resources (primarily disk space).



## DROP SUBSCRIPTION statement

<b>Function</b>	To drop a subscription for a user from a publication.
<b>Syntax</b>	<b>DROP SUBSCRIPTION TO</b> <i>publication-name</i> [ ( <i>string</i> ) ] ... <b>FOR</b> <i>userid</i> ,...
<b>Usage</b>	Anywhere. This statement is applicable only to SQL Remote.
<b>Permissions</b>	Must have DBA authority.
<b>Side effects</b>	Automatic commit.
<b>See also</b>	CREATE SUBSCRIPTION statement
<b>Description</b>	<p>Drops a SQL Remote subscription for a user ID to a publication in the current database. The user ID will no longer receive updates when data in the publication is changed.</p> <p>In SQL Remote, publications and subscriptions are two-way relationships. If you drop a subscription for a remote user to a publication on a consolidated database, you should also drop the subscription for the consolidated database on the remote database to prevent updates on the remote database being sent to the consolidated database.</p>
<b>Example</b>	<p>The following statement drops a subscription for the user ID <b>SamS</b> to the publication <b>pub_contact</b>.</p> <pre>DROP SUBSCRIPTION TO pub_contact FOR SamS</pre>

## EXECUTE statement

**Function** To execute a SQL statement.

**Syntax** Syntax 1

```
EXECUTE statement-name
... [ USING DESCRIPTOR sqlda-name
      | USING host-variable-list ]

... [ INTO DESCRIPTOR into-sqlda-name
      | INTO into-host-variable-list ]

... [ ARRAY :nnn ]
```

**Parameters** *statement-name*: *identifier*, or *host-variable*

*sqlda-name*: *identifier*

*into-sqlda-name*: *identifier*

Syntax 2

```
EXECUTE IMMEDIATE statement
```

*statement*: *string*, or *host-variable*

**Usage** Embedded SQL.

**Permissions** Permissions are checked on the statement being executed.

**Side effects** None.

**See also** PREPARE statement

DECLARE CURSOR statement

**Description** Format 1 executes the named dynamic statement which was previously prepared. If the dynamic statement contains host variable place holders which supply information for the request (bind variables), then either the **sqlda-name** must specify a C variable which is a pointer to an SQLDA containing enough descriptors for all bind variables occurring in the statement, or the bind variables must be supplied in the **host-variable-list**.

The optional ARRAY clause can be used with prepared INSERT statements, to allow wide inserts, which insert more than one row at a time and which may improve performance. The value **nnn** is the number of rows to be inserted. The SQLDA must contain  $nnn * (\text{columns per row})$  variables. The first row is placed in SQLDA variables 0 to  $(\text{columns per row})-1$ , and so on.

OUTPUT from a SELECT statement or a CALL statement is put either into the variables in the variable list or into the program data areas described by the named SQLDA. The correspondence is one to one from the OUTPUT (selection list or parameters) to either the host variable list or the SQLDA descriptor array.

If EXECUTE is used with an INSERT statement, the inserted row is returned in the second descriptor. For example, when using auto-increment primary keys or when using before insert triggers that generate primary key values, the EXECUTE statement provides a mechanism to re-fetch the row immediately and determine the primary key value assigned to the row. The same thing can be achieved by using @@identity with auto-increment keys.

Format 2 is a short form to PREPARE and EXECUTE a statement that does not contain bind variables or output. The SQL statement contained in the string or host-variable is immediately executed.

The EXECUTE statement can be used for any SQL statement that can be prepared. Cursors are used for SELECT statements or CALL statements that return many rows from the database (see "Cursors in Embedded SQL" in the chapter "The Embedded SQL Interface").

After successful execution of an INSERT, UPDATE or DELETE statement, the *sqlerrd[2]* field of the SQLCA (SQLCOUNT) is filled in with the number of rows affected by the operation.

## Examples

```

1. EXEC SQL EXECUTE IMMEDIATE
   'DELETE FROM employee WHERE emp_id = 105';

2. EXEC SQL PREPARE del_stmt FROM
   'DELETE FROM employee WHERE emp_id = :a';
   EXEC SQL EXECUTE del_stmt USING :employee_number;

3. EXEC SQL PREPARE sell FROM
   'SELECT emp_lname FROM employee WHERE emp_id = :a';
   EXEC SQL EXECUTE sell USING :employee_number INTO
   :emp_lname;

```

## EXECUTE IMMEDIATE statement

<b>Function</b>	To enable dynamically constructed statements to be executed from within a procedure.
<b>Syntax</b>	Syntax 1: <b>EXECUTE IMMEDIATE</b> <i>string-expression</i> Syntax 2: <b>EXECUTE</b> ( <i>string-expression</i> )  For information about the Embedded SQL EXECUTE IMMEDIATE statement, see "EXECUTE statement" on page 1062.
<b>Usage</b>	Procedures, triggers, and batches
<b>Permissions</b>	None. The statement is executed with the permissions of the owner of the procedure, not with the permissions of the user who calls the procedure.
<b>Side effects</b>	None. However, if the statement is a data definition statement with an automatic commit as a side effect, then that commit does take place.
<b>See also</b>	CREATE PROCEDURE statement Compound statements
<b>Description</b>	<p>The EXECUTE IMMEDIATE statement extends the range of statements that can be executed from within procedures and triggers. It allows dynamically prepared statements to be executed, such as statements that are constructed using the parameters passed in to a procedure.</p> <p>Literal strings in the statement must be enclosed in single quotes, and the statement must be on a single line.</p>
<b>Example</b>	<p>The following procedure creates a table, where the table name is supplied as a parameter to the procedure.</p> <p>The EXECUTE IMMEDIATE statement must all be on a single line if you run this example.</p> <pre>CREATE PROCEDURE CreateTableProc(     IN tablename char(30) ) BEGIN     EXECUTE IMMEDIATE 'CREATE TABLE '    tablename        ' ( column1 INT PRIMARY KEY)' END</pre> <p>To call the procedure and create a table <b>mytable</b>:</p> <pre>CALL CreateTableProc( 'mytable' )</pre>

## EXIT statement

<b>Function</b>	To leave ISQL.
<b>Syntax</b>	<b>EXIT   QUIT   BYE</b>
<b>Usage</b>	ISQL.
<b>Permissions</b>	None.
<b>Side effects</b>	Will do a commit if option COMMIT_ON_EXIT is ON (default); otherwise will do a rollback.
<b>See also</b>	SET OPTION statement
<b>Description</b>	Leave the ISQL environment and return to the operating system. This will close your connection with the database. Before doing so, ISQL will perform a COMMIT operation if the COMMIT_ON_EXIT option is ON. If the option is OFF, ISQL will perform a ROLLBACK. The default action is to COMMIT any changes you have made to the database.

## EXPLAIN statement

<b>Function</b>	To retrieve a text specification of the optimization strategy used for a particular cursor.
<b>Syntax</b>	<b>EXPLAIN PLAN FOR CURSOR</b> <i>cursor-name</i> <b>INTO</b> <i>host-variable</i> ... <b>INTO</b> <i>host-variable</i>   <b>USING DESCRIPTOR</b> <i>sqlda-name</i>
<b>Parameters</b>	<i>cursor-name</i> : <i>identifier, or host-variable</i> <i>sqlda-name</i> : <i>identifier</i>
<b>Usage</b>	Embedded SQL.
<b>Permissions</b>	Must have opened the named cursor.
<b>Side effects</b>	None.
<b>See also</b>	DECLARE CURSOR statement PREPARE statement FETCH statement CLOSE statement OPEN statement "Cursors in Embedded SQL" in the chapter "The Embedded SQL Interface" "The SQL communication area (SQLCA)" in the chapter "The Embedded SQL Interface"
<b>Description</b>	<p>The EXPLAIN statement retrieves a text representation of the optimization strategy for the named cursor. The cursor must be previously declared and opened.</p> <p>The <b>host-variable</b> or SQLDA variable must be of string type. The optimization string specifies in what order the tables are being searched and also which indexes are being used for the searches if any. This string can be quite long, but most optimization strings will fit into 300 characters.</p> <p>The format of this string is, in general:</p> <pre>table (index), table (index), ...</pre>

If a table has been given a correlation name, the correlation name will appear instead of the table name. The order that the table names appear in the list is the order in which they will be accessed by the database engine. After each table is a parenthesized index name. This is the index that will be used to access the table. If no index will be used (the table will be scanned sequentially) then the letters "seq" will appear for the index name. If a particular SQL SELECT statement involves subqueries, then a colon (:) will separate each subquery's optimization string. These subquery sections will appear in the order that the database engine will execute the queries.

After successful execution of the EXPLAIN statement, the **sqlerrd[3]** field of the SQLCA (SQLIOESTIMATE) will be filled in with an estimate of the number of input/output operations required to fetch all rows of the query.

A discussion with quite a few examples of the optimization string can be found in "Monitoring and Improving Performance".

### Examples

```
EXEC SQL BEGIN DECLARE SECTION;
char plan[300];
EXEC SQL END DECLARE SECTION;
EXEC SQL DECLARE employee_cursor CURSOR FOR
    SELECT empnum, empname
    FROM employee
    WHERE name like :pattern;
EXEC SQL OPEN employee_cursor;
EXEC SQL EXPLAIN PLAN FOR CURSOR employee_cursor
INTO :plan;
printf( "Optimization Strategy: '%s'.n", plan );
```

# FETCH statement

**Function** To reposition a cursor and then get data from it.

**Syntax**

```

FETCH
  {
    NEXT
  | PRIOR
  | FIRST
  | LAST
  | ABSOLUTE row-count
  | RELATIVE row-count
  }
  ... cursor-name
  ... [ INTO host-variable-list ]
        | USING DESCRIPTOR sqlda-name1
        | INTO variable-list
  ... [ PURGE ] [ BLOCK n ]
  ... [ FOR UPDATE ] [ ARRAY fetch-count ]

```

**Parameters**

*cursor-name*: *identifier*, or *host-variable*

*sqlda-name*: *identifier*

*host-variable-list*: may contain indicator variables

*row-count*: *number* or *host variable*

*fetch-count*: *integer* or *host variable*

**Usage** Embedded SQL, procedures, triggers, and batches..

The following clauses are for use in Embedded SQL only:

- ◆ **USING DESCRIPTOR** *sqlda-name*
- ◆ **INTO** *host-variable-list*
- ◆ **PURGE**
- ◆ **BLOCK** *n*
- ◆ **ARRAY** *fetch-count*
- ◆ Use of *host-variable* in *cursor-name* and *row-count*.

The **INTO** *variable-list* clause is for use in procedures and triggers only:

**Permissions** The cursor must be opened and the user must have **SELECT** permission on the tables referenced in the declaration of the cursor.

**Side effects** None.



**See also**

DECLARE CURSOR statement  
PREPARE statement  
OPEN statement  
"Cursors in Embedded SQL" in the chapter "The Embedded SQL Interface"  
"Using cursors in procedures and triggers" in the chapter "Using Procedures, Triggers, and Batches"

**Description**

The FETCH statement retrieves one row from the named cursor.

The ARRAY clause allows so-called **wide fetches**, which retrieve more than one row at a time, and which may improve performance.

The cursor must have been previously opened.

One or more rows from the result of the SELECT statement is put either into the variables in the variable list or into the program data areas described by the named SQLDA. In either case, the correspondence is one to one from the select list to either the host variable list or the SQLDA descriptor array.

The INTO clause is optional. If it is not specified, then the FETCH statement positions the cursor only (see the following paragraphs).

An optional positional parameter can be specified that allows the cursor to be moved before a row is fetched. The default is NEXT which causes the cursor to be advanced one row before the row is fetched. PRIOR causes the cursor to be backed up one row before fetching.

RELATIVE positioning is used to move the cursor by a specified number of rows in either direction before fetching. A positive number indicates moving forward and a negative number indicates moving backwards. Thus, a NEXT is equivalent to RELATIVE 1 and PRIOR is equivalent to RELATIVE -1. RELATIVE 0 retrieves the same row as the last fetch statement on this cursor.

The ABSOLUTE positioning parameter is used to go to a particular row. A zero indicates the position before the first row (see "Using cursors in procedures and triggers" in the chapter "Using Procedures, Triggers, and Batches").

A one (1) indicates the first row, and so on. Negative numbers are used to specify an absolute position from the end of the cursor. A negative one (-1) indicates the last row of the cursor. FIRST is a short form for ABSOLUTE 1. LAST is a short form for ABSOLUTE -1.

The OPEN statement initially positions the cursor before the first row.

If the fetch includes a positioning parameter and the position is outside the allowable cursor positions, then the SQLE\_NOTFOUND warning is issued.

**Cursor positioning problems**

Inserts and some updates to DYNAMIC SCROLL cursors can cause problems with cursor positioning. The database engine will not put inserted rows at a predictable position within a cursor unless there is an ORDER BY clause on the SELECT statement. In some cases, the inserted row will not appear at all until the cursor is closed and opened again.

With SQL Anywhere, this occurs if a temporary table had to be created to open the cursor (see "Temporary tables used in query processing" in the chapter "Monitoring and Improving Performance" for a description).

The UPDATE statement may cause a row to move in the cursor.

This will happen if the cursor has an ORDER BY that uses an existing index (a temporary table is not created).

The FOR UPDATE clause indicates that the fetched row will subsequently be updated with an UPDATE WHERE CURRENT OF CURSOR statement. This clause causes the database engine to put a write lock on the row. The lock will be held until the end of the current transaction. See "How locking works" in the chapter "Using Transactions and Locks".

**Using the FETCH statement in Embedded SQL**

The DECLARE CURSOR statement must appear before the FETCH statement in the C source code, and the OPEN statement must be executed before the FETCH statement. If a host variable is being used for the cursor name, then the DECLARE statement actually generates code and thus must be executed before the FETCH statement.

In the multi-user environment, rows may be fetched by the client more than one at a time. Note that in QNX, the client is linked into the application so this will always happen by default. This is referred to as block fetching or multi-row fetching. The first fetch causes several rows to be sent back from the server. The client buffers these rows and subsequent fetches are retrieved from these buffers without a new request to the server. The BLOCK clause gives the client and server a hint as to how many rows may be fetched by the application. The special value of 0 means the request will be sent to the server and a single row will be returned (no row blocking). The PURGE clause causes the client to flush its buffers of all rows and then send the fetch request to the server. Note that this fetch request may return a block of rows.

If the `SQLSTATE_NOTFOUND` warning is returned on the fetch, then the `sqlerrd[2]` field of the `SQLCA (SQLCOUNT)` will contain the number of rows that the attempted fetch exceeded the allowable cursor positions. (A cursor can be on a row, before the first row or after the last row.) The value is 0 if the row was not found but the position is valid, for example, executing `FETCH RELATIVE 1` when positioned on the last row of a cursor. The value will be positive if the attempted fetch was further beyond the end of the cursor, and negative if the attempted fetch was further before the beginning of the cursor.

After successful execution of the fetch statement, the `sqlerrd[1]` field of the `SQLCA (SQLIOCOUNT)` will be incremented by the number of input/output operations required to perform the fetch. This field is actually incremented on every database statement.

To use wide fetches in Embedded SQL, include the fetch statement in your code as follows:

```
EXEC SQL FETCH . . . ARRAY nnn
```

where `ARRAY nnn` is the last item of the `FETCH` statement. The fetch count `nnn` can be a host variable. The `SQLDA` must contain `nnn * (columns per row)` variables. The first row is placed in `SQLDA` variables 0 to `(columns per row)-1`, and so on.

The engine returns in `SQLCOUNT` the number of records fetched and always returns a `SQLCOUNT` greater than zero unless there is an error. Older versions of the engine or server only return a single row and the `SQLCOUNT` is set to zero. Thus a `SQLCOUNT` of zero with no error condition indicates one valid row has been fetched.

### Embedded SQL Example

```
EXEC SQL DECLARE cur_employee CURSOR FOR
SELECT emp_id, emp_lname FROM employee ;
EXEC SQL OPEN cur_employee;
EXEC SQL FETCH cur_employee
INTO :emp_number, :emp_name:indicator;
```

For a detailed example of using wide fetches, see the section "Fetching more than one row at a time" in the chapter "The Embedded SQL Interface".

### Procedure/trigger Example

```
BEGIN
DECLARE cur_employee CURSOR FOR
    SELECT emp_lname
    FROM employee ;
DECLARE name CHAR(40) ;
OPEN cur_employee;
LOOP
FETCH NEXT cur_employee into name ;
    . . .
END LOOP
CLOSE cur_employee;
```

END

LOOP statement, DECLARE CURSOR statement, FETCH statement,  
LEAVE statement

# FOR statement

<b>Function</b>	Repeat the execution of a statement list once for each row in a cursor.
<b>Syntax</b>	<pre>[ <i>statement-label</i> : ] ...FOR <i>for-loop-name</i> AS <i>cursor-name</i> ...  CURSOR FOR <i>statement</i> ...  [ FOR UPDATE   FOR READ ONLY ] ...  DO <i>statement-list</i> ...  END FOR [ <i>statement-label</i> ]</pre>
<b>Usage</b>	Procedures, triggers, and batches only.
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>See also</b>	DECLARE CURSOR statement FETCH statement LEAVE statement LOOP statement
<b>Description</b>	<p>The FOR statement is a control statement that allows you to execute a list of SQL statements once for each row in a cursor. The FOR statement is equivalent to a compound statement with a DECLARE for the cursor and a DECLARE of a variable for each column in the result set of the cursor followed by a loop that fetches one row from the cursor into the local variables and executes <i>statement-list</i> once for each row in the cursor.</p> <p>The name and data type of the local variables that are declared are derived from the <i>statement</i> used in the cursor. With a SELECT statement, the data type will be the data type of the expressions in the select list. The names will be the select list item aliases where they exist; otherwise, they will be the name of the columns. Any select list item that is not a simple column reference must have an alias. With a CALL statement, the names and data types will be taken from the RESULT clause in the procedure definition.</p> <p>The LEAVE statement can be used to resume execution at the first statement after the END FOR. If the ending <i>statement-label</i> is specified, it must match the beginning <i>statement-label</i>.</p>
<b>Example</b>	<pre>FOR names AS curs CURSOR FOR SELECT emp_lname FROM employee DO CALL search_for_name( emp_lname ); END FOR;</pre>

# FROM clause

**Function** To specify the database tables or views involved in a SELECT or UPDATE statement.

**Syntax** ... **FROM** *table-expression*, ...

**Parameters** *table-expression*:  
    *table-spec*  
    | *table-expression* *join-type* *table-spec* [ **ON** *condition* ]  
    | ( *table-expression*, ... )

*table-spec*:  
    [ *userid* . ] *table-name* [ [**AS**] *correlation-name* ]

*join-type*:  
    **CROSS JOIN**  
    | [ **NATURAL** | **KEY** ] **JOIN**  
    | [ **NATURAL** | **KEY** ] **INNER JOIN**  
    | [ **NATURAL** | **KEY** ] **LEFT OUTER JOIN**  
    | [ **NATURAL** | **KEY** ] **RIGHT OUTER JOIN**

**Usage** Anywhere.

**Permissions** Must be connected to the database.

**Side effects** None.

**See also** SELECT statement  
UPDATE statement  
DELETE statement  
"Search conditions" in the chapter "Watcom-SQL Language Reference"

**Description** The SELECT and UPDATE statements require a table list to specify which tables will be used by the statement.

**Views**

Although this description refers to tables, it applies to views unless otherwise noted.

The FROM table list creates a result set consisting of all the columns from all the tables specified. Initially, all combinations of rows in the component tables are in the result set, and the number of combinations is usually reduced by **join** conditions and/or WHERE conditions.

Tables owned by a different user can be qualified by specifying the user ID. Tables owned by groups to which the current user belongs will be found by default without specifying the user ID (see "Referring to tables owned by groups" in the chapter "Managing User IDs and Permissions").

The **correlation name** is used to give a temporary name to the table for this SQL statement only. This is useful when referencing columns which must be qualified by a table name but the table name is long and cumbersome to type. The correlation name is also necessary to distinguish between table instances when referencing the same table more than once in the same query. If no correlation name is specified, then the table name is used as the correlation name for the current statement.

If the same correlation name is used twice for the same table in a table expression, that table is treated as if it were only listed once. For example, in

```
SELECT *
FROM sales_order
KEY JOIN sales_order_items,
sales_order
KEY JOIN employee
```

the two instances of the `sales_order` table are treated as one instance, and is equivalent to

```
SELECT *
FROM sales_order_items
KEY JOIN sales_order
KEY JOIN employee
```

whereas

```
SELECT *
FROM Person HUSBAND, Person WIFE
```

would be treated as two instances of the `Person` table, with different correlation names `HUSBAND` and `WIFE`.

## Joining tables

A `JOIN` reduces the result set based on the **join type** and **join condition**. The join types are described below and the join condition is specified after the keyword `ON`.

Parentheses can also be used to join one table to more than one other table. For example:

```
A JOIN (B,C)
```

joins table `A` to both tables `B` and `C`.

Table expressions can be arbitrarily complex. For example,

```
A JOIN B JOIN C
A JOIN ( B, C JOIN D )
```

are legal and meaningful table list expressions. (Any of the valid join types could have been used in the above examples.)

## Cross joins

A CROSS JOIN does not restrict the results of the join. The query

```
SELECT *
FROM table1
CROSS JOIN table2
```

has a result set as follows:

- ◆ As long as table1 is not the same name as table2:
  - ◆ The columns of the result set includes all columns in table1 and also all columns in table2.
  - ◆ There is one row in the result set for each combination of a row in table1 and a row in table2. If table1 has  $n1$  rows and table2 has  $n2$  rows, the query returns  $n1 * n2$  rows.
- ◆ If table1 is the same table as table2, and neither is given a correlation name, the result set is simply the rows of table1.

## Generated join conditions

Natural joins and key joins are **generated join conditions**: that is, the keyword KEY or NATURAL indicates a restriction on the join results.

For a natural join, the generated join condition is based on the names of columns in the tables being joined; for a key join, the condition is based on a foreign key relationship between the two tables.

## Natural joins

A NATURAL JOIN restricts the results by comparing the values of columns in the two tables with the same column name. An error is reported if there are no common column names. A join condition can optionally be specified which further restricts the results of the join.

Column names such as Description or Address often cause a NATURAL JOIN to return different results than expected.



## Key joins

A **KEY JOIN** restricts the result set based on a foreign key relationship between the two tables. A join condition can optionally be specified which further restricts the results of the join. A key join is valid if exactly one foreign key is identified between the two tables; otherwise, an error indicating the ambiguity is reported.

Parentheses can also be used to join one table to more than one other table. For example

```
A KEY JOIN (B,C)
```

joins table A to both tables B and C.

A join involving parentheses is valid if there is an unambiguous join for each table listed in the parentheses.

A **KEY JOIN** with a view is valid if there is a valid **KEY JOIN** with exactly one of the tables in the **FROM** table list of the view definition.

There are two conditions where the meaning of a **KEY JOIN** is ambiguous. In the first condition, there are two tables A and B where A has a foreign key for B and B has a foreign key for A. The second condition occurs when a table A has two foreign keys for a table B. In either case, the primary table *must* have a correlation name which is the same as the **role name** of the foreign key. Otherwise an error will be reported. For example, every SQL Anywhere database has a table called **SYSTABLEPERM** to hold permission information. Every row in the permission table has two foreign keys for the table describing user IDs (**SYSUSERPERM**). One foreign key is for the user ID giving the permission (role name **grantor**) and the other is the user ID getting the permission (role name **grantee**). A **KEY JOIN** for the **grantor** would look like the following:

```
SYSUSERPERM grantor
KEY JOIN SYSTABLEPERM
```

and a **KEY JOIN** for both would look like:

```
SYSUSERPERM grantor
KEY JOIN SYSTABLEPERM
KEY JOIN SYSUSERPERM grantee
```

## INNER JOIN and OUTER JOIN

All joins described thus far have been **INNER JOINS**, which is the default. Each row of

```
customer INNER JOIN sales_order
```

contains the information from one customer row and one sales\_order row. If a particular customer has placed no orders, there will be no information for that customer.

```
A LEFT OUTER JOIN B
```

includes all rows of table A whether or not there is a row in B that satisfies the join condition. If a particular row in A has no matching row in B, the columns in the join corresponding to table B will contain the NULL value. Similarly,

```
A RIGHT OUTER JOIN B
```

includes all rows of table B whether or not there is a row in A that satisfies the join condition.

## Join conditions

A **join condition** can be specified for any join type except CROSS JOIN. The simplest form of join condition is to use it instead of using a KEY or NATURAL join. In the sample database, the following are equivalent:

```
SELECT *
FROM sales_order
JOIN customer
ON sales_order.cust_id = customer.id
SELECT *
FROM sales_order
KEY JOIN customer
```

The following two are also equivalent:

```
SELECT *
FROM department
JOIN employee
ON department.dept_id = employee.dept_id
SELECT *
FROM department
NATURAL JOIN employee
```

With INNER joins, specifying a join condition is equivalent to adding the join condition to the WHERE clause. However, this is not true for OUTER joins.

A join condition on an OUTER join is part of the join operation. For example, the statement:

```
SELECT *
FROM employee
KEY LEFT OUTER JOIN Skill
ON skill_name = 'COBOL'
```

produces a table containing all employees whether or not they have the skill COBOL. Those who do not have the skill COBOL will have the NULL value in the Skill columns. On the other hand, the query:

```
SELECT *
FROM employee
KEY LEFT OUTER JOIN Skill
WHERE skill_name = 'COBOL'
```

can be thought of as two separate stages.

- 1 The first stage creates the table specified by the FROM clause: SELECT \* FROM employee KEY LEFT OUTER JOIN Skill which has at least one row for each employee. Employees who do not have the skill COBOL will have the NULL value in the Skill table columns. For all other employees, there will be one row for each of their skills.
- 2 The second stage takes this result and applies the condition:

```
WHERE skill_name = 'COBOL'
```

which removes employees without any skills (since the condition is UNKNOWN), and only keeps the skill COBOL for the other employees. Thus, the result has no rows with the NULL value in the Skill columns, so it is clearly different from the result achieved using the join condition.

OUTER joins with join conditions can be complicated. If you are having problems using a join condition, try removing the WHERE clause from the statement to verify that the join is retrieving the rows you expected.

## Join abbreviations

SQL Anywhere provides the following abbreviations for joins.

**table1 [INNER | LEFT OUTER | RIGHT OUTER] JOIN table2** If there is no ON **join condition** specified, the join is assumed to be a KEY join. All key joins are a SQL Anywhere extension. Note that KEY JOIN is *not* assumed if a join condition is specified.

**table1 JOIN table2 ON join-condition** The default join type is an INNER JOIN.

### Examples

The following are valid FROM clauses:

```
... FROM employee
... FROM employee NATURAL JOIN department
... FROM customer
   KEY JOIN sales_order
```

*FROM clause*

---

```
KEY JOIN sales_order_items  
KEY JOIN product
```

# GET DATA statement

<b>Function</b>	To get string or binary data for one column on the current row of a cursor. GET DATA is usually used to fetch LONG BINARY or LONG VARCHAR fields. See "SET statement" for putting long values into the database.
<b>Syntax</b>	<pre> <b>GET DATA</b> <i>cursor-name</i> <b>COLUMN</b> <i>column-num</i> <b>OFFSET</b> <i>start-offset</i> ... [ <b>WITH TEXTPTR</b> ] ... <b>USING DESCRIPTOR</b> <i>sqlda-name</i>   <b>INTO</b> <i>host-variable</i> [, ... ]   </pre>
<b>Parameters</b>	<p><i>cursor-name</i>: <i>identifier</i>, or <i>host-variable</i></p> <p><i>column-num</i>: <i>integer</i> or <i>host-variable</i></p> <p><i>start-offset</i>: <i>integer</i> or <i>host-variable</i></p> <p><i>sqlda-name</i>: <i>identifier</i></p>
<b>Usage</b>	Embedded SQL.
<b>Permissions</b>	The cursor must be opened and positioned on a row using FETCH.
<b>Side effects</b>	None.
<b>See also</b>	<p>FETCH statement</p> <p>"Transact-SQL READTEXT statement" in the chapter "Using Transact-SQL with SQL Anywhere"</p> <p>"Transact-SQL WRITETEXT statement" in the chapter "Using Transact-SQL with SQL Anywhere"</p>
<b>Description</b>	<p>Get a piece of one column value from the row at the current cursor position. The value of <i>column-num</i> starts at one and identifies which column's data is to be fetched. That column must be of a string or binary type.</p> <p>The <i>start-offset</i> indicates the number of bytes to skip over in the field value. Normally, this would be the number of bytes previously fetched. The number of bytes fetched on this GET DATA statement is determined by the length of the target host variable.</p> <p>The indicator value for the target host variable is a short integer, so it cannot always contain the number of bytes truncated. Instead, it contains a negative value if the field contains the NULL value, and a positive value (NOT necessarily the number of bytes truncated) if the value is truncated, and zero if a non-NULL value is not truncated.</p> <p>If the WITH TEXTPTR clause is given, a text pointer is retrieved into a second indicator variable or into the second field in the SQLDA. This text pointer can be used with the Transact-SQL READ TEXT and WRITE TEXT statements.</p>

The total length of the data is returned in the SQLCOUNT field of the SQLCA structure.

### Example

The following example uses GET DATA to fetch a **binary large object** (often called a **blob**).

```
EXEC SQL BEGIN DECLARE SECTION;
DECL_BINARY(1000) piece;
short ind;
long offset;
EXEC SQL END DECLARE SECTION;
int size;
/* Open a cursor on a long varchar field */
EXEC SQL DECLARE big_cursor CURSOR FOR
SELECT long_data FROM some_table
WHERE key_id = 2;
EXEC SQL OPEN big_cursor;
EXEC SQL FETCH big_cursor INTO :piece;
for( offset = 0; ; offset += piece.len ) {
    EXEC SQL GET DATA big_cursor COLUMN 1
    OFFSET :offset INTO :piece:ind;
    /* Done if the NULL value */
    if( ind < 0 ) break;
    write_out_piece( piece );
    /* Done when the piece was not truncated */
    if( ind == 0 ) break;
}
EXEC SQL CLOSE big_cursor;
```

# GET DESCRIPTOR statement

<b>Function</b>	Retrieves information about variables within a descriptor area, or retrieves actual data from a variable in a descriptor area
<b>Syntax</b>	<b>GET DESCRIPTOR</b> <i>descriptor-name</i> ...{ <i>hostvar</i> = <b>COUNT</b> }   <b>VALUE</b> <i>n</i> <i>assignment</i> [,...] }
<b>Parameters</b>	<i>assignment</i> : <i>hostvar</i> = { <b>TYPE</b>   <b>LENGTH</b>   <b>PRECISION</b>   <b>SCALE</b>   <b>DATA</b>   <b>INDICATOR</b>   <b>NAME</b>   <b>NULLABLE</b>   <b>RETURNED_LENGTH</b> }
<b>Usage</b>	Embedded SQL.
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>See also</b>	ALLOCATE DESCRIPTOR statement DEALLOCATE DESCRIPTOR statement SET DESCRIPTOR statement "The SQL descriptor area (SQLDA)" in the chapter "The Embedded SQL Interface"
<b>Description</b>	<p>The Get descriptor statement is used to retrieve information about variables within a descriptor area, or to retrieve actual data from a variable in a descriptor area.</p> <p>The value <i>n</i> specifies the variable in the descriptor area about which the information will be retrieved. Type checking is performed when doing GET ... DATA to ensure that the host variable and the descriptor variable have the same data type.</p> <p>If an error occurs, it is returned in the SQLCA.</p>
<b>Example</b>	For an example, see "ALLOCATE DESCRIPTOR statement" on page 970.

## GET OPTION statement

<b>Function</b>	To find the current setting of an option.
<b>Syntax</b>	<b>GET OPTION</b> [ <i>userid</i> . ] <i>option-name</i> ...   <b>INTO</b> <i>host-variable</i>   <b>USING DESCRIPTOR</b> <i>sqlda-name</i>
<b>Parameters</b>	<i>userid</i> : <i>identifier, string or host-variable</i> <i>sqlda-name</i> : <i>identifier</i> <i>option-name</i> : <i>identifier, string or host-variable</i> <i>host-variable</i> :     indicator variable allowed
<b>Usage</b>	Embedded SQL.
<b>Permissions</b>	Must have DBA authority to get someone else's option settings.
<b>Side effects</b>	None.
<b>See also</b>	SET OPTION statement
<b>Description</b>	<p>The GET OPTION statement gets the option setting of the option <i>option-name</i> for the user <i>userid</i> or for the connected user if <i>userid</i> is not specified. This will be either the user's personal setting or the <b>PUBLIC</b> setting if there is no setting for the connected user. If the option specified is a database option and the user has a temporary setting for that option, then the temporary setting is retrieved.</p> <p>If <i>option-name</i> does not exist, GET OPTION returns the warning <b>SQLE_NOTFOUND</b>.</p>
<b>Example</b>	<pre>EXEC SQL GET OPTION 'date_format' INTO :datefmt;</pre>



# GRANT statement

**Function** To give permissions to specific users and to create new user IDs.

**Syntax** Syntax 1:  
**GRANT CONNECT TO** *userid*,... **IDENTIFIED BY** *password*,...

Syntax 2:  
**GRANT**  
     **DBA**  
     | **RESOURCE**  
     | **GROUP**  
     | **MEMBERSHIP IN GROUP** *userid*,...  
 ... **TO** *userid*,...

Syntax 3:  
**GRANT**  
     [  
       **ALTER**  
       | **DELETE**  
       | **INSERT**  
       | **REFERENCES** [ ( *column-name*,... ) ]  
       | **SELECT** [ ( *column-name*,... ) ]  
       | **UPDATE** [ ( *column-name*,... ) ]  
       | **ALL [ PRIVILEGES ]**  
     ], ...  
 ... **ON** [ *owner*.]*table-name* **TO** *userid*, ... [ **WITH GRANT OPTION** ]

Syntax 4:  
**GRANT EXECUTE ON** [ *owner*.]*procedure-name* **TO** *userid*,...

**Usage** Anywhere.

**Permissions** For Syntax 1 or 2, must:

- ◆ Be changing your own password using GRANT CONNECT
- ◆ Be adding members to your own user ID, or
- ◆ Have DBA authority.

If changing another user's password, the other user must not be connected to the database.

For Syntax 3, must have one of the following:

- ◆ Created the table

- ◆ Been granted permissions on the table with GRANT OPTION
- ◆ DBA authority

For Syntax 4, must have one of the following:

- ◆ Created the procedure
- ◆ DBA authority

**Side effects**

Automatic commit.

**See also**

REVOKE statement

**Description**

The GRANT statement is used to grant database permissions to individual user IDs and groups. It is also used to create and delete users and groups.

Syntax 1 and 2 of the GRANT statement are used for granting special privileges to users as follows:

**CONNECT** Creates a new user. GRANT CONNECT can also be used by any user to change their own password. To create a user with the empty string as the password, type:

```
GRANT CONNECT TO userid IDENTIFIED BY ""
```

To create a user with no password, type:

```
GRANT CONNECT TO userid
```

A user with no password cannot connect to the database. This is useful when creating groups when you do not want anyone to connect to the group user ID. The password must be a valid identifier, as described in "Watcom-SQL language elements" in the chapter "Watcom-SQL Language Reference".

**DBA** Database Administrator authority gives a user permission to do anything. This is usually reserved for the person in the organization who is looking after the database.

**RESOURCE** Allows the user to create tables and views.

**GROUP** Allows the user(s) to have members. See "Managing groups" in the chapter "Managing User IDs and Permissions" for a complete description.

**MEMBERSHIP IN GROUP userid,...** This allows the user(s) to inherit table permissions from a group and to reference tables created by the group without qualifying the table name. See "Managing groups" in the chapter "Managing User IDs and Permissions" for a complete description.

Syntax 3 of the GRANT statement is used to grant permission on individual tables or views. The table permissions can be listed together, or specifying ALL grants all six permissions at once. The permissions have the following meaning:

**ALTER** The users will be allowed to alter this table with the ALTER TABLE statement. This permission is not allowed for views.

**DELETE** The users will be allowed to delete rows from this table or view.

**INSERT** The users will be allowed to insert rows into the named table or view.

**REFERENCES [(column-name,...)]** The users will be allowed to create indexes on the named tables, and foreign keys which reference the named tables. If column names are specified, then the users will be allowed to reference only those columns. REFERENCES permissions on columns cannot be granted for views, only for tables.

**SELECT [(column-name,...)]** The users will be allowed to look at information in this view or table. If column names are specified, then the users will be allowed to look at only those columns. SELECT permissions on columns cannot be granted for views, only for tables.

**UPDATE [(column-name,...)]** The users will be allowed to update rows in this view or table. If column names are specified, then the users will be allowed to update only those columns. UPDATE permissions on columns cannot be granted for views, only for tables.

**ALL** All of the above permissions.

With Release 5.0.02, there is a change in permission requirements for statements that require reading of a column value. A statement such as

```
UPDATE table_name
SET col_name = col_name + 1
```

or

```
UPDATE table_name
SET col_name = 1
WHERE col_name = 2
```

both require that information in the column be read, and so require both UPDATE and SELECT permissions on the column col\_name. Prior to release 5.0.02, only UPDATE permissions were required to carry out operations of this sort.

If WITH GRANT OPTION is specified, then the named user ID is also given permission to GRANT the same permissions to other user IDs.

Format 4 of the GRANT statement is used to grant permission to execute a procedure.

**Examples**

Make two new users for the database.

```
GRANT
CONNECT TO Laurel, Hardy
IDENTIFIED BY Stan, Ollie
```

Grant permissions on the employee table to user Laurel.

```
GRANT
SELECT, UPDATE ( street )
ON employee
TO Laurel
```

Allow the user Hardy to execute the Calculate\_Report procedure.

```
GRANT
EXECUTE ON Calculate_Report
TO Hardy
```

## GRANT CONSOLIDATE statement

<b>Function</b>	To identify the database immediately above the current database in a SQL Remote hierarchy, who will receive messages from the current database.
<b>Syntax</b>	<pre>GRANT CONSOLIDATE TO <i>userid</i>, ... ... TYPE <i>message-system</i>, ... ... ADDRESS <i>address-string</i>, ... ... [ SEND { EVERY   AT } '<i>hh:mm</i>' ]</pre>
<b>Parameters</b>	<p><i>message-system</i>:</p> <p><b>MAP   FILE   VIM   SMTP</b></p>
<b>Usage</b>	Anywhere. This statement is applicable only to SQL Remote.
<b>Permissions</b>	Must have DBA authority.
<b>Side effects</b>	Automatic commit.
<b>See also</b>	<p>GRANT REMOTE statement</p> <p>REVOKE CONSOLIDATE statement</p> <p>GRANT PUBLISH statement</p> <p>"Granting and revoking REMOTE and CONSOLIDATE permissions" in the chapter "SQL Remote Administration",</p>
<b>Description</b>	<p>In a SQL Remote installation, the database immediately above the current database in a SQL Remote hierarchy must be granted CONSOLIDATE permissions. GRANT CONSOLIDATE is issued at a remote database to identify its consolidated database. Each database can have only one user ID with CONSOLIDATE permissions: you cannot have a database that is a remote database for more than one consolidated database.</p> <p>The consolidated user is identified by a message system, identifying the method by which messages are sent to and received from the consolidated user. The currently supported message systems are MAPI and FILE. The VIM system is expected to be in beta by the time of release. The address-name must be a valid address for the message-system, enclosed in single quotes.</p> <p>For the FILE message type, the address is a subdirectory of the directory pointed to by the SQLREMOTE environment variable.</p>

The GRANT CONSOLIDATE statement is required for the consolidated database to receive messages, but does not by itself subscribe the consolidated database to any data. To subscribe to data, a subscription must be created for the consolidated user ID to one of the publications in the current database. Running the database extraction utility at a consolidated database creates a remote database with the proper GRANT CONSOLIDATE statement already issued.

The optional SEND EVERY and SEND AT clauses specify a frequency at which messages are sent. The string contains a time that is a length of time between messages (for SEND EVERY) or a time of day at which messages are sent (for SEND AT). With SEND AT, messages are sent once per day.

If a user has been granted remote permissions without a SEND EVERY or SEND AT clause, the Message Agent processes messages, and then stops. In order to run the Message Agent continuously, you must ensure that every user with REMOTE permission has either a SEND AT or SEND EVERY frequency specified.

It is anticipated that at many remote databases, the Message Agent will be run periodically, and that the consolidated database will have no SEND clause specified.

### **Example**

```
GRANT CONSOLIDATE TO con_db
TYPE mapi
ADDRESS 'Consolidated Database'
```

# GRANT PUBLISH statement

<b>Function</b>	To identify the publisher of the current database.
<b>Syntax</b>	<b>GRANT PUBLISH TO</b> <i>userid</i>
<b>Usage</b>	Anywhere. This statement is applicable only to SQL Remote.
<b>Permissions</b>	Must have DBA authority.
<b>Side effects</b>	Automatic commit.
<b>See also</b>	GRANT REMOTE statement GRANT CONSOLIDATE statement REVOKE PUBLISH statement CREATE PUBLICATION statement CREATE SUBSCRIPTION statement

**Description** Each database in a SQL Remote installation is identified in outgoing messages by a user ID, called the **publisher**. The GRANT PUBLISH statement identifies the publisher user ID associated with these outgoing messages.

Only one user ID can have PUBLISH authority. The user ID with PUBLISH authority is identified by the special constant CURRENT PUBLISHER. The following query identifies the current publisher:

```
SELECT CURRENT PUBLISHER
```

If there is no publisher, the special constant is NULL.

The current publisher special constant can be used as a default setting for columns. It is often useful to have a CURRENT PUBLISHER column as part of the primary key for replicating tables, as this helps prevent primary key conflicts due to updates at more than one site.

In order to change the publisher, you must first drop the current publisher using the REVOKE PUBLISH statement, and then create a new publisher using the GRANT PUBLISH statement.

**Example**

```
GRANT PUBLISH TO publisher_ID
```

# GRANT REMOTE statement

<b>Function</b>	To identify a database immediately below the current database in a SQL Remote hierarchy, who will receive messages from the current database. These are called remote users.
<b>Syntax</b>	<b>GRANT REMOTE TO</b> <i>userid</i> , ... ... <b>TYPE</b> <i>message-system</i> , ... ... <b>ADDRESS</b> <i>address-string</i> , ... ... [ <b>SEND</b> { <b>EVERY</b>   <b>AT</b> } ' <i>hh:mm</i> ' ]
<b>Parameters</b>	<i>message-system</i> : <b>MAP</b>   <b>FILE</b>   <b>VIM</b>   <b>SMTP</b>
<b>Usage</b>	Anywhere. This statement is applicable only to SQL Remote.
<b>Permissions</b>	Must have DBA authority.
<b>Side effects</b>	Automatic commit.
<b>See also</b>	GRANT CONSOLIDATE statement REVOKE REMOTE statement GRANT PUBLISH statement "Granting and revoking REMOTE and CONSOLIDATE permissions" in the chapter "SQL Remote Administration",
<b>Description</b>	<p>In a SQL Remote installation, each database receiving messages from the current database must be granted REMOTE permissions.</p> <p>The single exception is the database immediately above the current database in a SQL Remote hierarchy, which must be granted CONSOLIDATE permissions.</p> <p>The remote user is identified by a message system, identifying the method by which messages are sent to and received from the consolidated user. The currently supported message systems are MAPI and FILE. The VIM message system is expected to be in beta at the time of release. The address-name must be a valid address for the message-system, enclosed in single quotes.</p> <p>For the FILE message type, the address is a subdirectory of the directory pointed to by the SQLREMOTE environment variable.</p> <p>The GRANT REMOTE statement is required for the remote database to receive messages, but does not by itself subscribe the remote user to any data. To subscribe to data, a subscription must be created for the user ID to one of the publications in the current database, using the database extraction utility or the CREATE SUBSCRIPTION statement.</p>



The optional `SEND EVERY` and `SEND AT` clauses specify a frequency at which messages are sent. The string contains a time that is a length of time between messages (for `SEND EVERY`) or a time of day at which messages are sent (for `SEND AT`). With `SEND AT`, messages are sent once per day.

If a user has been granted remote permissions without a `SEND EVERY` or `SEND AT` clause, the Message Agent processes messages, and then stops. In order to run the Message Agent continuously, you must ensure that every user with `REMOTE` permission has either a `SEND AT` or `SEND EVERY` frequency specified.

It is anticipated that at many consolidated databases, the Message Agent will be run continuously, so that all remote databases would have a `SEND` clause specified. A typical setup may involve sending messages to laptop users daily (`SEND AT`) and to remote servers every hour or two (`SEND EVERY`). You should use as few different times as possible, for efficiency.

**Example**

```
GRANT REMOTE TO SamS
TYPE mapi
ADDRESS 'Singer, Samuel'
```

## HELP statement

<b>Function</b>	To receive help in the ISQL environment.
<b>Syntax</b>	<b>HELP</b> [ <i>topic</i> ]
<b>Usage</b>	ISQL.
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>Description</b>	The HELP statement is used to enter the ISQL interactive help facility. The <i>topic</i> for help can be optionally specified. If <i>topic</i> is not specified, then the help system is entered at the index.

# IF statement

<b>Function</b>	Provide conditional execution of SQL statements.
<b>Syntax</b>	<pre> <b>IF</b> <i>search-condition</i> <b>THEN</b> <i>statement-list</i> ... [ <b>ELSEIF</b> <i>search-condition</i> <b>THEN</b> <i>statement-list</i> ] ... ... [ <b>ELSE</b> <i>statement-list</i> ] ... <b>END IF</b> </pre>
<b>Usage</b>	Procedures, triggers, and batches.
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>See also</b>	Compound statements The chapter "Using Procedures, Triggers, and Batches"
<b>Description</b>	The IF statement is a control statement that allows you to conditionally execute the first list of SQL statements whose <i>search-condition</i> evaluates to TRUE. If no <i>search-condition</i> evaluates to TRUE, and an ELSE clause exists, the <i>statement-list</i> in the ELSE clause is executed. Execution resumes at the first statement after the END IF.
<b>Example</b>	<pre> CREATE PROCEDURE TopCustomer (OUT TopCompany CHAR(35),                                OUT TopValue INT) BEGIN     DECLARE err_notfound EXCEPTION     FOR SQLSTATE '02000' ;     DECLARE curThisCust CURSOR FOR     SELECT company_name, CAST(     sum(sales_order_items.quantity *     product.unit_price) AS INTEGER) VALUE     FROM customer     LEFT OUTER JOIN sales_order     LEFT OUTER JOIN sales_order_items     LEFT OUTER JOIN product     GROUP BY company_name ;      DECLARE ThisValue INT ;     DECLARE ThisCompany CHAR(35) ;     SET TopValue = 0 ;     OPEN curThisCust ;     CustomerLoop:     LOOP         FETCH NEXT curThisCust         INTO ThisCompany, ThisValue ;         IF SQLSTATE = err_notfound THEN             LEAVE CustomerLoop ;         END IF ;         IF ThisValue &gt; TopValue THEN </pre>

## *IF statement*

---

```
        SET TopValue = ThisValue ;
        SET TopCompany = ThisCompany ;
    END IF ;
END LOOP CustomerLoop ;
CLOSE curThisCust ;
END
```

# INCLUDE statement

<b>Function</b>	Include a file into a source program to be scanned by the SQL source language preprocessor.
<b>Syntax</b>	<b>INCLUDE</b> <i>filename</i>
<b>Parameters</b>	<i>filename:</i> <i>identifier</i>
<b>Usage</b>	Embedded SQL.
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>Description</b>	<p>The INCLUDE statement is very much like the C preprocessor <b>#include</b> directive. However, the SQL preprocessor will read the given file inserting its contents into the output C file. Thus, if an include file contains information that the SQL preprocessor requires, it should be included with the Embedded SQL INCLUDE statement.</p> <p>Two file names are specially recognized: SQLCA and SQLDA. Any C program using Embedded SQL must contain an</p> <pre>EXEC SQL INCLUDE SQLCA;</pre> <p>statement before any Embedded SQL statements. This statement must appear at a position in the C program where static variable declarations are allowed. Many Embedded SQL statements require variables (invisible to the programmer) which are declared by the SQL preprocessor at the position of the SQLCA include statement. The SQLDA file must be included if any SQLDA's are used.</p>

# INPUT statement

<b>Function</b>	To import data into a database table from an external file or from the keyboard.
<b>Syntax</b>	<pre><b>INPUT INTO</b> [ <i>owner.</i>]<i>table-name</i> ... [ <b>FROM</b> <i>filename</i>   <b>PROMPT</b> ] ... [ <b>FORMAT</b> <i>input-format</i> ] ... [ <b>ESCAPE CHARACTER</b> <i>character</i> ] ... [ <b>BY ORDER</b>   <b>BY NAME</b> ] ... [ <b>DELIMITED BY</b> <i>string</i> ] ... [ <b>COLUMN WIDTHS</b> (<i>integer</i>,...) ] ... [ <b>NOSTRIP</b> ] ... [ ( <i>column-name</i>, ... ) ]</pre>
<b>Usage</b>	ISQL.
<b>Permissions</b>	Must have INSERT permission on the table or view.
<b>Side effects</b>	None.
<b>See also</b>	OUTPUT statement INSERT statement UPDATE statement DELETE statement SET OPTION statement
<b>Description</b>	<p>The INPUT statement allows efficient mass insertion into a database table. Lines of input are read either from the user via an input window (if PROMPT is specified) or from a file (if FROM filename is specified). If neither is specified, the input will be read from the command file containing the input statement—this can even be directly from the ISQL editor. In this case, input is ended with a line containing only the string END.</p> <p>These lines are inserted into the named table. If a column list is specified, the data is inserted into the specified columns of the named table.</p> <p>Normally, the INPUT statement stops when it attempts to insert a row that causes an error. Errors can be treated in different ways by setting the ON_ERROR and CONVERSION_ERROR options (see SET OPTION). ISQL will print a warning in the statistics window if any string values are truncated on INPUT. Missing values for NOT NULL columns will be set to zero for numeric types and to the empty string for non-numeric types.</p> <p>The default escape character for characters stored as hexadecimal codes and symbols is a backslash (\), so that \x0A is the linefeed character, for example.</p>

The escape character can be changed using the ESCAPE CHARACTER clause. For example, to use the exclamation mark as the escape character, you would enter

```
... ESCAPE CHARACTER '!'
```

Only one single-byte character can be used as an escape character.

The BY clause allows the user to specify whether the columns from the input file should be matched up with the table columns based on their ordinal position in the lists (ORDER, the default) or by their names (NAME). Not all input formats have column name information in the file. NAME is allowed only for those formats that do. They are the same formats that allow automatic table creation listed below: DBASEII, DBASEIII, DIF, FOXPRO, LOTUS, and WATFILE.

The DELIMITED BY clause allows you to specify a string to be used as the delimiter in ASCII input format.

COLUMN WIDTHS can be specified for FIXED format only. It specifies the widths of the columns in the input file. If COLUMN WIDTHS is not specified, then the widths are determined by the database column types.

Normally, for ASCII input format, trailing blanks will be stripped from quoted strings before the value is inserted. NOSTRIP can be used to suppress trailing blank stripping. Trailing blanks are not stripped from quoted strings, regardless of whether the option is used.

If the ASCII file has entries such that a column appears to be null, LOAD TABLE treats it as null. If the column in that position cannot be null, inserts a zero in numeric columns and an empty string in character columns.

Each set of values must occupy one input line and must be in the format specified by the FORMAT clause or the format set by the SET INPUT\_FORMAT statement if the format clause is not specified. When input is entered by the user, an empty screen is provided for the user to enter one row per line in the input format.

Certain file formats contain information about column names and types. Using this information, the INPUT statement will create the database table if it does not already exist. This is a very easy way to load data into the database. The formats that have enough information to create the table are: DBASEII, DBASEIII, DIF, FOXPRO, LOTUS, and WATFILE.

Allowable input formats are:

**ASCII** Input lines are assumed to be ASCII characters, one row per line, with values separated by commas. Alphabetic strings may be enclosed in apostrophes (single quotes) or quotation marks (double quotes). Strings containing commas must be enclosed in either single or double quotes. If single or double quotes are used, double the quote character to use it within the string. Optionally, you can use the **DELIMITED BY** clause to specify a different delimiter string than the default which is a comma.

Three other special sequences are also recognized. The two characters `.n` represent a newline character, `\` represents a single `,` and the sequence `.xDD` represents the character with hexadecimal code `DD`.

**DBASE** The file is in dBASE II or dBASE III format. ISQL will attempt to determine which of the two dBase formats the file is based on information in the file. If the table doesn't exist, it will be created.

**DBASEII** The file is in dBASE II format. If the table doesn't exist, it will be created.

**DBASEIII** The file is in dBASE III format. If the table doesn't exist, it will be created.

**DIF** Input file is in Data Interchange Format. If the table doesn't exist, it will be created.

**FIXED** Input lines are in fixed format. The width of the columns can be specified using the **COLUMN WIDTHS** clause. If they are not specified, then column widths in the file must be the same as the maximum number of characters required by any value of the corresponding database column's type.

**FOXPRO** The file is in FoxPro format (the FoxPro memo field is different than the dBASE memo field). If the table doesn't exist, it will be created.

**LOTUS** The file is a Lotus WKS format worksheet. **INPUT** assumes that the first row in the Lotus WKS format worksheet is column names. If the table doesn't exist, it will be created. In this case, the types and sizes of the columns created may not be correct because the information in the file pertains to a cell, not to a column.

**WATFILE** The input will be a WATFILE file. If the table doesn't exist, it will be created.

Input from a command file is terminated by a line containing **END**. Input from a file is terminated at the end of the file.

**Example**

The following is an example of an **INPUT** statement from an ASCII text file.

```
INPUT INTO employee
```



```
FROM new_emp.inp  
FORMAT ascii;
```

# INSERT statement

**Function** To insert a single row (format 1) or a selection of rows from elsewhere in the database (format 2) into a table.

**Syntax** Format 1  
**INSERT INTO** [ *owner.*]*table-name* [( *column-name*, ... )]  
... **VALUES** ( *expression* | **DEFAULT**, ... )

Format 2  
**INSERT INTO** [ *owner.*]*table-name* [( *column-name*, ... )]  
... *select-statement*

**Usage** Anywhere.

**Permissions** Must have INSERT permission on the table.

**Side effects** None.

**See also** INPUT statement  
UPDATE statement  
DELETE statement  
PUT statement

**Description** The INSERT statement is used to add new rows to a database table.

Format 1 allows the insertion of a single row with the specified expression values. The keyword DEFAULT can be used to cause the default value for the column to be inserted. If the optional list of column names is given, the values are inserted one for one into the specified columns. If the list of column names is not specified, the values are inserted into the table columns in the order they were created (the same order as retrieved with SELECT \*). The row is inserted into the table at an arbitrary position. (In relational databases, tables are not ordered.)

Format 2 allows the user to do mass insertion into a table with the results of a fully general SELECT statement. Insertions are done in an arbitrary order unless the SELECT statement contains an ORDER BY clause. The columns from the select list are matched ordinarily with the columns specified in the column list or the columns in the order they were created.

**Note**

The NUMBER(\*) function is useful for generating primary keys with format 2 of the INSERT statement (see "Watcom-SQL Functions").

Inserts can be done into views provided the SELECT statement defining the view has only one table in the FROM clause and does not contain a GROUP BY clause, an aggregate function, or involve a UNION operation.

Character strings inserted into tables are always stored in the case they are entered, regardless of whether the database is case sensitive or not. Thus a string *Value* inserted into a table is always held in the database with an upper-case V and the remainder of the letters lower case. SELECT statements return the string as *Value*. If the database is not case-sensitive, however, all comparisons make *Value* the same as *value*, *VALUE*, and so on. Further, if a single-column primary key already contains an entry *Value*, an INSERT of *value* is rejected, as it would make the primary key not unique.

#### Performance hint

To insert many rows into a table, it is more efficient to declare a cursor and use the PUT statement to insert the rows, where possible, than to carry out many separate INSERT statements.

#### Examples

Add an Eastern Sales department to the database.

```
INSERT
INTO department ( dept_id, dept_name )
VALUES ( 230, 'Eastern Sales' )
```

Fill the table dept\_head with the names of department heads and their departments.

```
INSERT
INTO dept_head (name, dept)
SELECT emp_fname || ' ' || emp_fname
      AS name,
      dept_name
FROM employee JOIN department
ON emp_id = dept_head_id
```

## LEAVE statement

<b>Function</b>	Continue execution by leaving a compound statement or LOOP.
<b>Syntax</b>	<b>LEAVE</b> <i>statement-label</i>
<b>Usage</b>	Procedures, triggers, and batches.
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>See also</b>	LOOP statement FOR statement Compound statements The chapter "Using Procedures, Triggers, and Batches"
<b>Description</b>	<p>The LEAVE statement is a control statement that allows you to leave a labeled compound statement or a labelled loop. Execution resumes at the first statement after the compound statement or loop.</p> <p>The compound statement that is the body of a procedure or triggers has an implicit label that is the same as the name of the procedure or trigger.</p>
<b>Examples</b>	<p>The following fragment shows how the LEAVE statement is used to leave a loop.</p>

```
SET i = 1;
lbl:
LOOP
  INSERT
  INTO Counters ( number )
  VALUES ( i ) ;
  IF i >= 10 THEN
    LEAVE lbl ;
  END IF ;
  SET i = i + 1
END LOOP lbl
```

The following example fragment uses LEAVE in a nested loop.

```
outer_loop:
LOOP
  SET i = 1;
  inner_loop:
  LOOP
    ...
    SET i = i + 1;
    IF i >= 10 THEN
      LEAVE outer_loop
    END IF
  END LOOP inner_loop
END LOOP outer_loop
```

# LOAD TABLE statement

<b>Function</b>	To import data into a database table from an external ascii-format file.
<b>Syntax</b>	<pre>LOAD [ INTO ] TABLE [ owner ].table-name ... FROM 'filename-string' ... [ FORMAT 'ascii' ] ... [ DELIMITED BY string ] ... [ STRIP { ON   OFF } ] ... [ QUOTES { ON   OFF } ] ... [ ESCAPES { ON   OFF } ] ... [ ESCAPE CHARACTER character ]</pre>
<b>Usage</b>	Anywhere.
<b>Permissions</b>	Must be the creator of the table or have DBA authority. Requires an exclusive lock on the table.
<b>Side effects</b>	Triggers, including referential integrity actions, are not fired by the LOAD TABLE statement. A COMMIT is performed at the end of the load.
<b>See also</b>	UNLOAD TABLE statement INPUT statement
<b>Description</b>	<p>The LOAD TABLE statement allows efficient mass insertion into a database table from an ASCII file. LOAD TABLE is more efficient than the ISQL statement INPUT and can be called from any client application.</p> <p>LOAD TABLE places an exclusive lock on the whole table; it does not fire any triggers associated with the table.</p> <p>You can use LOAD TABLE on a temporary table, but the temporary table must have been declared with the ON COMMIT PRESERVE ROWS clause, as LOAD TABLE does a COMMIT after the load.</p> <p>If the ASCII file has entries such that a column appears to be null, LOAD TABLE treats it as null. If the column in that position cannot be null, inserts a zero in numeric columns and an empty string in character columns.</p> <p>The following list describes each of the clauses of the statement.</p> <p><b>Filename-string</b> The filename-string is passed to the engine as a string. The string is therefore subject to the same formatting requirements as other SQL strings. In particular:</p> <ul style="list-style-type: none"> <li>◆ To indicate directory paths, the backslash character \ must be represented by two backslashes. Therefore, the statement to load data from the file C:\TEMP\INPUT.DAT into the employee table is:</li> </ul>

```
LOAD TABLE employee FROM 'c:\\temp\\input.dat' ...
```

- ◆ The pathname is relative to the database engine, not to the client application. If you are running the statement on a database server on some other computer, the directory names refers to directories on the server machine, not on the client machine.
- ◆ You can use UNC path names to load data from files on computers other than the server. For example, on a Windows for Workgroups, Windows 95, or Windows NT network you may use the following statement to load data from a file on the client machine:

```
LOAD TABLE employee FROM  
'\\\\client\\temp\\input.dat'
```

**FORMAT option** The only file format currently supported is ASCII. Input lines are assumed to be ASCII characters, one row per line, with values separated by the column delimiter character.

**DELIMITED BY option** The default column delimiter character is a comma. You can specify an alternative column delimiter by providing a string. Only the first ASCII character of the string is read. The same formatting requirements apply as to other SQL strings. In particular, to specify tab-delimited values the hexadecimal ASCII code of the tab character (9) is used. The DELIMITED BY clause is as follows:

```
...DELIMITED BY '\\x09' ...
```

**STRIP option** With STRIP turned on (the default), trailing blanks are stripped from values before they are inserted. To turn the STRIP option off, the clause is as follows:

```
...STRIP OFF ...
```

Trailing blanks are stripped only for non-quoted strings. Quoted strings retain their trailing blanks.

**QUOTES option** With QUOTES turned on (the default), the LOAD statement looks for a quote character. The quote character is either an apostrophe (single quote) or a quotation mark (double quote). The first such character encountered in the input file is treated as the quote character for the input file.

With quotes on, column delimiter characters can be included in column values. Also, quote characters are assumed not to be part of the value. Therefore, a line of the form

```
'123 High Street, Anytown', (715) 398-2354
```

is treated as two values, not three, despite the presence of the comma in the address. Also, the quotes surrounding the address are not inserted into the database.

To include a quote character in a value, with `QUOTES` on, you must use two quotes. The following line includes a value in the third column that is a single quote character:

```
'123 High Street, Anytown', '(715)398-2354', '''
```

**ESCAPES option** With `ESCAPES` turned on (the default), characters following the backslash character are recognized and interpreted as special characters by the database engine. New line characters can be included as the combination `\n`, other characters can be included in data as hexadecimal ASCII codes, such as `\x09` for the tab character. A sequence of two backslash characters (`\\`) is interpreted as a single backslash.

**ESCAPE CHARACTER option** The default escape character for characters stored as hexadecimal codes and symbols is a backslash (`\`), so that `\x0A` is the linefeed character, for example.

This can be changed using the `ESCAPE CHARACTER` clause. For example, to use the exclamation mark as the escape character, you would enter

```
... ESCAPE CHARACTER '!'
```

Only one single-byte character can be used as an escape character.

## LOOP statement

<b>Function</b>	Repeat the execution of a statement list.
<b>Syntax</b>	[ <i>statement-label</i> : ] ...[ <b>WHILE</b> <i>search-condition</i> ] <b>LOOP</b> ... <i>statement-list</i> ... <b>END LOOP</b> [ <i>statement-label</i> ]
<b>Usage</b>	Procedures, triggers, and batches.
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>See also</b>	FOR statement LEAVE statement
<b>Description</b>	The WHILE and LOOP statements are control statements that allow you to repeatedly execute a list of SQL statements while a <i>search-condition</i> evaluates to TRUE. The LEAVE statement can be used to resume execution at the first statement after the END LOOP.  If the ending <i>statement-label</i> is specified, it must match the beginning <i>statement-label</i> .

**Examples**                   A While loop in a procedure.

```

...
SET i = 1 ;
WHILE i <= 10 LOOP
    INSERT INTO Counters( number ) VALUES ( i ) ;
    SET i = i + 1 ;
END LOOP ;
...

```

A labeled loop in a procedure.

```

SET i = 1;
lbl:
LOOP
    INSERT
    INTO Counters( number )
    VALUES ( i ) ;
    IF i >= 10 THEN
        LEAVE lbl ;
    END IF ;
    SET i = i + 1 ;
END LOOP lbl

```



# MESSAGE statement

<b>Function</b>	To display a message on the message window of the database engine or server.
<b>Syntax</b>	<b>MESSAGE</b> <i>expression</i>
<b>Usage</b>	Procedures, triggers, and batches.
<b>Permissions</b>	Must be connected to the database.
<b>Side effects</b>	None.
<b>See also</b>	CREATE PROCEDURE statement
<b>Description</b>	The MESSAGE statement displays an expression on the message window of the database engine or server (not on the client window). It is used primarily for debugging of procedures and triggers.
<b>Example</b>	The following procedure displays a message on the engine message window:

```
CREATE PROCEDURE message_test ()  
BEGIN  
MESSAGE 'procedure called successfully' ;  
END
```

## The statement

```
CALL message_test()
```

displays the string *procedure called successfully* on the database engine message window.

# NULL value

<b>Function</b>	To specify a value that is unknown or not applicable.
<b>Syntax</b>	<b>NULL</b>
<b>Usage</b>	Anywhere.
<b>Permissions</b>	Must be connected to the database.
<b>Side effects</b>	None.
<b>See also</b>	"Expressions" in the chapter "Watcom-SQL Language Reference" "Search conditions" in the chapter "Watcom-SQL Language Reference"
<b>Description</b>	The NULL value is a special value which is different from any valid value for any data type. However, the NULL value is a legal value in any data type. The NULL value is used to represent missing or inapplicable information. Note that these are two separate and distinct cases where NULL is used:

Situation	Description
missing	The field does have a value, but that value is unknown.
inapplicable	The field does not apply for this particular row.

SQL allows columns to be created with the NOT NULL restriction. This means that those particular columns cannot contain the NULL value.

The NULL value introduces the concept of three valued logic to SQL. The NULL value compared using any comparison operator with any value including the NULL value is "UNKNOWN." The only search condition that returns "TRUE" is the IS NULL predicate. In SQL, rows are selected only if the search condition in the WHERE clause evaluates to TRUE; rows that evaluate to UNKNOWN or FALSE are not selected.

The IS [ NOT ] *truth-value* clause, where *truth-value* is one of TRUE, FALSE or UNKNOWN can also be used to select rows where the NULL value is involved. See "Search conditions" in the chapter "Watcom-SQL Language Reference" for a description of this clause.

In the following examples, the column Salary contains the NULL value.

Condition	Truth value	Selected?
Salary = NULL	UNKNOWN	NO
Salary <> NULL	UNKNOWN	NO

Condition	Truth value	Selected?
NOT (Salary = NULL)	UNKNOWN	NO
NOT (Salary <> NULL)	UNKNOWN	NO
Salary = 1000	UNKNOWN	NO
Salary IS NULL	TRUE	YES
Salary IS NOT NULL	FALSE	NO
Salary = 1000 IS UNKNOWN	TRUE	YES

The same rules apply when comparing columns from two different tables. Therefore, joining two tables together will not select rows where any of the columns compared contain the NULL value.

The NULL value also has an interesting property when used in numeric expressions. The result of *any* numeric expression involving the NULL value is the NULL value. This means that if the NULL value is added to a number, the result is the NULL value—not a number. If you want the NULL value to be treated as 0, you must use the ISNULL( *expression*, 0 ) function (see "Watcom-SQL Functions").

Many common errors in formulating SQL queries are caused by the behavior of NULL. You will have to be careful to avoid these problem areas. See "Search conditions" in the chapter "Watcom-SQL Language Reference" for a description of the effect of three-valued logic when combining search conditions.

### Example

The following INSERT statement inserts a NULL into the date\_returned column of the Borrowed\_book table.

```
INSERT
INTO Borrowed_book
( date_borrowed, date_returned, book )
VALUES ( CURRENT DATE, NULL, '1234' )
```

# OPEN statement

<b>Function</b>	To open a previously declared cursor to access information from the database.
<b>Syntax</b>	<b>OPEN</b> <i>cursor-name</i> ... [ <b>USING DESCRIPTOR</b> <i>sqlda-name</i> ]   <b>USING</b> <i>host-variable</i> , ...   ... [ <b>WITH HOLD</b> ] ... [ <b>ISOLATION LEVEL</b> <i>n</i> ] ... [ <b>BLOCK</b> <i>n</i> ]
<b>Parameters</b>	<i>cursor-name</i> : <i>identifier, or host-variable</i> <i>sqlda-name</i> : <i>identifier</i>
<b>Usage</b>	Embedded SQL, procedures, triggers, and batches.  The USING DESCRIPTOR <b>sqlda-name</b> , <b>host-variable</b> and <b>BLOCK n</b> formats are for Embedded SQL only.
<b>Permissions</b>	Must have SELECT permission on all tables in a SELECT statement or EXECUTE permission on the procedure in a CALL statement.  When the cursor is on a CALL statement, OPEN causes the procedure to execute until the first result set (SELECT statement with no INTO clause) is encountered. If the procedure completes and no result set is found, the SQLSTATE_PROCEDURE_COMPLETE warning is set.
<b>Side effects</b>	None.
<b>See also</b>	DECLARE CURSOR statement PREPARE statement FETCH statement RESUME statement CLOSE statement
<b>Description</b>	The OPEN statement opens the named cursor. The cursor must be previously declared.  By default, all cursors are automatically closed at the end of the current transaction (COMMIT or ROLLBACK executed). The optional WITH HOLD clause will keep the cursor open for subsequent transactions. It will remain open until the end of the current connection or until an explicit CLOSE statement is executed. Cursors are automatically closed when a connection is terminated.

The ISOLATION LEVEL clause allows this cursor to be opened at an isolation level different from the current setting of the ISOLATION\_LEVEL option. All operations on this cursor will be performed at the specified isolation level regardless of the option setting. If this clause is not specified, then the cursor's isolation level for the entire time the cursor is open is the value of the ISOLATION\_LEVEL option when the cursor is opened. See "How locking works" in the chapter "Using Transactions and Locks".

The cursor is positioned before the first row (see "Cursors in Embedded SQL" in the chapter "The Embedded SQL Interface" or "Using cursors in procedures and triggers" in the chapter "Using Procedures, Triggers, and Batches").

## Embedded SQL

If the cursor name is specified by an identifier or string, then the corresponding DECLARE CURSOR statement must appear prior to the OPEN in the C program; if the cursor name is specified by a host variable, then the DECLARE cursor statement must execute before the OPEN statement.

The optional USING clause specifies the host variables that will be bound to the place-holder bind variables in the SELECT statement for which the cursor has been declared.

The multiuser support fetches rows in blocks (more than 1 at a time). By default, the number of rows in a block is determined dynamically based on the size of the rows and how long it takes the database engine to fetch each row. The application can specify a maximum number of rows that should be contained in a block by specifying the BLOCK clause. For example, if you are fetching and displaying 5 rows at a time, use **BLOCK 5**. Specifying **BLOCK 0** will cause 1 record at a time to be fetched and also cause a FETCH RELATIVE 0 to always fetch the row again.

After successful execution of the OPEN statement, the *sqlerrd[3]* field of the SQLCA (SQLIOESTIMATE) will be filled in with an estimate of the number of input/output operations required to fetch all rows of the query. Also, the *sqlerrd[2]* field of the SQLCA (SQLCOUNT) will be filled in with either the actual number of rows in the cursor (a value greater than or equal to 0), or an estimate thereof (a negative number whose absolute value is the estimate). It will be the actual number of rows if the database engine can compute it without counting the rows. The database can also be configured to always return the actual number of rows (see the ROW\_COUNTS option in SET OPTION statement.), but this can be expensive.

## Examples

The following examples show the use of OPEN in Embedded SQL.

```
1. EXEC SQL OPEN employee_cursor;
2. EXEC SQL PREPARE emp_stat FROM
'SELECT empnum, empname FROM employee WHERE name
like ?';
EXEC SQL DECLARE employee_cursor CURSOR FOR
emp_stat;
EXEC SQL OPEN employee_cursor USING :pattern;

BEGIN
DECLARE cur_employee CURSOR FOR
    SELECT emp_lname
    FROM employee ;
DECLARE name CHAR(40) ;
OPEN cur_employee;
LOOP
FETCH NEXT cur_employee into name ;
    ...
END LOOP
CLOSE cur_employee;
END
```

**Procedure/trigger  
Example**

# OUTPUT statement

<b>Function</b>	To output the current query results to a file.
<b>Syntax</b>	<pre>OUTPUT TO <i>filename</i> ... [ <b>FORMAT</b> <i>output_format</i> ] ... [ <b>ESCAPE CHARACTER</b> <i>character</i> ] ... [ <b>DELIMITED BY</b> <i>string</i> ] ... [ <b>QUOTE</b> <i>string</i> [ <b>ALL</b> ] ] ... [ <b>COLUMN WIDTHS</b> (<i>integer</i>,...) ]</pre>
<b>Usage</b>	ISQL.
<b>Permissions</b>	None.
<b>Side effects</b>	The current query results that are displayed in the ISQL data window are repositioned to the top.
<b>See also</b>	SELECT statement

## Description

The OUTPUT statement copies the information retrieved by the **current query** to *filename*. The output format can be specified with the optional format clause. If no format clause is specified, the OUTPUT\_FORMAT option setting is used (see "SET OPTION statement" on page 1149).

The **current query** is the SELECT or INPUT statement which generated the information displayed in the ISQL data window. The OUTPUT statement will report an error if there is no current query.

The default escape character for characters stored as hexadecimal codes and symbols is a backslash (\), so that \x0A is the linefeed character, for example.

This can be changed using the ESCAPE CHARACTER clause. For example, to use the exclamation mark as the escape character, you would enter

```
... ESCAPE CHARACTER ' ! '
```

The DELIMITED BY and QUOTE clauses are for the ASCII output format only. The delimiter string will be placed between columns (default comma) and the quote string will be placed around string values (default '—single quote). If ALL is specified in the QUOTE clause, then the quote string will be placed around all values, not just strings.

The COLUMN WIDTH clause is used to specify the column widths for the FIXED format output.

Allowable output formats are:

**ASCII** The output is an ASCII format file with one row per line in the file. All values are separated by commas and strings are enclosed in apostrophes (single quotes). The delimiter and quote strings can be changed using the **DELIMITED BY** and **QUOTE** clauses. If **ALL** is specified in the **QUOTE** clause, then all values (not just strings) will be quoted.

Three other special sequences are also used. The two characters `\n` represent a newline character, `\\` represents a single backslash, and the sequence `\xDD` represents the character with hexadecimal code `DD`. This is the default output format.

**DBASEII** The output is a dBASE II format file with the column definitions at the top of the file. Note that a maximum of 32 columns can be output. Also, note that columns longer than 255 characters will be truncated in the file.

**DBASEIII** The output is a dBASE III format file with the column definitions at the top of the file. Note that a maximum of 128 columns can be output. Also, note that columns longer than 255 characters will be truncated in the file.

**DIF** The output is a file in the standard Data Interchange Format.

**FIXED** The output is fixed format with each column having a fixed width. The width for each column can be specified using the **COLUMN WIDTH** clause. If this clause is omitted, the width for each column is computed from the data type for the column, and is large enough to hold any value of that data type. No column headings are output in this format.

**FOXPRO** The output is a FoxPro format file (the FoxPro memo field is different than the dBASE memo field) with the column definitions at the top of the file. Note that a maximum of 128 columns can be output. Also, note that columns longer than 255 characters will be truncated in the file.

**LOTUS** The output is a Lotus WKS format worksheet. Column names will be put as the first row in the worksheet. Note that there are certain restrictions on the maximum size of Lotus WKS format worksheets that other software (such as Lotus 1-2-3) can load. There is no limit to the size of file ISQL can produce.

**SQL** The output is an ISQL INPUT statement required to recreate the information in the table.

**TEXT** The output is a TEXT format file which prints the results in columns with the column names at the top and vertical lines separating the columns. This format is similar to that used to display data in the ISQL data window.



**WATFILE** The output is a WATFILE format file with the column definitions at the top of the file.

**Example**

Place the contents of the employee table in a file, in ASCII format.

```
SELECT *  
FROM employee ;  
OUTPUT TO employee.txt  
FORMAT ASCII
```

## PARAMETERS statement

<b>Function</b>	To specify parameters to an ISQL command file.
<b>Syntax</b>	<b>PARAMETERS</b> <i>parameter1, parameter2, ...</i>
<b>Usage</b>	ISQL.
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>See also</b>	READ statement "Command Files"
<b>Description</b>	<p>The PARAMETERS statement specifies how many parameters there are to a command file and also gives names to those parameters so that they can be referenced later in the command file.</p> <p>Parameters are referenced by putting:</p> <pre>{parameter1}</pre> <p>into the file where you wish the named parameter to be substituted. There must be no spaces between the braces and the parameter name.</p> <p>If a command file is invoked with fewer than the required number of parameters, ISQL prompts for values of the missing parameters.</p> <p>See "Command Files".</p>
<b>Example</b>	<p>The following ISQL command file takes two parameters.</p> <pre>PARAMETERS department_id, file ; SELECT emp_lname FROM employee WHERE dept_id = {department_id} &gt;#{file}.dat;</pre>

## PASSTHROUGH statement

<b>Function</b>	To start or stop passthrough mode for SQL Remote administration. Forms 1 and 2 start passthrough mode, while form 3 stops passthrough mode.
<b>Syntax</b>	<p>Syntax 1.       <b>PASSTHROUGH [ ONLY ] FOR <i>userid</i>,...</b></p> <p>Syntax 2.       <b>PASSTHROUGH [ ONLY ] FOR SUBSCRIPTION</b>                          ... <b>TO [ ( <i>owner</i> ) ],<i>publication-name</i> [ ( <i>constant</i> ) ]</b></p> <p>Syntax 3.       <b>PASSTHROUGH STOP</b></p>
<b>Usage</b>	Anywhere. This statement is applicable only to SQL Remote.
<b>Permissions</b>	Must have DBA authority.
<b>Side effects</b>	None.
<b>Description</b>	<p>In passthrough mode, any SQL statements are executed by the database server, and are also placed into the transaction log to be sent in messages to subscribers. If the ONLY keyword is used to start passthrough mode, the statements are not executed at the server; they are sent to recipients only. The recipients of the passthrough SQL statements are either a list of user IDs (form 1) or all subscribers to a given publication. Passthrough mode may be used to apply changes to a remote database from the consolidated database or send statements from a remote database to the consolidated database.</p> <p>Syntax 2 sends statements to remote databases whose subscriptions are started, and does not send statements to remote databases whose subscriptions are created and not started.</p>

### Example

```

PASSTHROUGH FOR rem_db ;
...
( SQL statements to be executed at the remote
  database )
...
PASSTHROUGH STOP ;

```

# PREPARE statement

<b>Function</b>	To prepare a statement to be executed later or used for a cursor.
<b>Syntax</b>	<pre>PREPARE <i>statement-name</i>       FROM <i>statement</i>       ...[ DESCRIBE <i>describe-type</i> INTO [ [ SQL ] DESCRIPTOR ] <i>descriptor</i>       ]       ...[ WITH EXECUTE ]</pre>
<b>Parameters</b>	<p><i>statement-name</i>: <i>identifier</i>, or <i>host-variable</i></p> <p><i>statement</i> : <i>string</i>, or <i>host-variable</i></p> <p><i>describe-type</i>:</p> <pre>{ ALL   BIND VARIABLES   INPUT   OUTPUT   SELECT LIST } ...[       LONG NAMES [ [ OWNER. ]TABLE. ]COLUMN ]         WITH VARIABLE RESULT       ]</pre>
<b>Usage</b>	Embedded SQL.
<b>Permissions</b>	None.
<b>Side effects</b>	Any statement previously prepared with the same name is lost.
<b>See also</b>	DECLARE CURSOR statement DESCRIBE statement OPEN statement EXECUTE statement DROP STATEMENT statement
<b>Description</b>	<p>The PREPARE statement prepares a SQL statement from the <b>statement</b> and associates the prepared statement with <b>statement-name</b>. This statement name is referenced to execute the statement, or to open a cursor if the statement is a SELECT statement. <b>Statement-name</b> may be a host variable of type <b>a_sql_statement_number</b> defined in the <b>sqlca.h</b> header file that is automatically included. If an identifier is used for the <b>statement-name</b>, then only one statement per module may be prepared with this <b>statement-name</b>.</p> <p>If a host variable is used for <b>statement-name</b>, it must have the type <b>short int</b>. There is a typedef for this type in <b>sqlca.h</b> called <b>a_sql_statement_number</b>. This type is recognized by the SQL preprocessor and can be used in a declare section. The host variable is filled in by the database during the PREPARE statement and need not be initialized by the programmer.</p>

If the DESCRIBE INTO DESCRIPTOR clause is used, the prepared statement is described into the specified descriptor. The describe type may be any of the describe types allowed in the DESCRIBE statement.

If the WITH EXECUTE clause is used, the statement is executed if and only if it is not a CALL or SELECT statement, and it has no host variables. The statement is immediately dropped after a successful execution. If the prepare and the describe (if any) are successful but the statement cannot be executed, a warning SQLCODE 111, SQLSTATE 01W08 is set, and the statement is not dropped.

The DESCRIBE INTO DESCRIPTOR and WITH EXECUTE clauses may improve performance, as they cut down on the required client/server communication.

#### Describing variable result sets

The WITH VARIABLE RESULT clause is used to describe procedures that may have more than one result set, with different numbers or types of columns.

If WITH VARIABLE RESULT is used, the database engine sets the SQLCOUNT value after the describe to one of the following values:

- ◆ **0** The result set may change: the procedure call should be described again following each OPEN statement.
- ◆ **1** The result set is fixed. No redescrbing is required.

#### Statements that can be prepared

The following is a list of statements that can be PREPARED.

- ◆ ALTER
- ◆ CALL
- ◆ COMMENT ON
- ◆ CREATE
- ◆ DELETE
- ◆ DROP
- ◆ GRANT
- ◆ INSERT
- ◆ REVOKE
- ◆ SELECT
- ◆ SET OPTION
- ◆ UPDATE
- ◆ VALIDATE TABLE.

**Compatibility issue**

For compatibility reasons, preparing COMMIT, PREPARE TO COMMIT, and ROLLBACK statements is still supported. However, we recommend that you do all transaction management operations with static Embedded SQL because certain application environments may require it. Also, other Embedded SQL systems do not support dynamic transaction management operations.

**Drop statement after use**

You should make sure that you DROP the statement after use. If you do not, then the memory associated with the statement is not reclaimed.

**Example**

```
EXEC SQL PREPARE employee_statement FROM  
'SELECT * from employee';
```

## PREPARE TO COMMIT statement

<b>Function</b>	To check whether a COMMIT can be performed.
<b>Syntax</b>	<b>PREPARE TO COMMIT</b>
<b>Usage</b>	Anywhere.
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>See also</b>	COMMIT statement ROLLBACK statement
<b>Description</b>	The PREPARE TO COMMIT statement tests whether a COMMIT can be performed successfully. The statement will cause an error if a COMMIT is not possible without violating the integrity of the database.

**Examples**

```
EXECUTE IMMEDIATE "SET OPTION wait_for_commit =
'on'";
EXECUTE IMMEDIATE "DELETE FROM employee
WHERE emp_id = 160";
EXECUTE IMMEDIATE "PREPARE TO COMMIT";
```

The following sequence of statements allows the delete to take place, even though it causes integrity violations. The PREPARE TO COMMIT statement causes an error reporting the violation.

```
SET OPTION wait_for_commit= 'ON' ;
DELETE
FROM department
WHERE dept_id = 100 ;
PREPARE TO COMMIT ;
```

## PUT statement

<b>Function</b>	To insert a row into the table(s) specified by the cursor. See "SET statement" for putting LONG VARCHAR or LONG BINARY values into the database.
<b>Syntax</b>	<pre><b>PUT</b> <i>cursor-name</i> ... [ <b>USING DESCRIPTOR</b> <i>sqlda-name</i>         <b>FROM</b> <i>host-variable-list</i> ]  ... [ <b>INTO DESCRIPTOR</b> <i>into-sqlda-name</i>         <b>INTO</b> <i>into-host-variable-list</i> ]  ... [ <b>ARRAY</b> :<i>nnn</i> ]</pre>
<b>Parameters</b>	<p><i>cursor-name</i>: <i>identifier</i>, or <i>host-variable</i></p> <p><i>sqlda-name</i>: <i>identifier</i></p> <p><i>host-variable-list</i>: may contain indicator variables</p>
<b>Usage</b>	Embedded SQL.
<b>Permissions</b>	Must have INSERT permission.
<b>Side effects</b>	None.
<b>See also</b>	UPDATE statement UPDATE (positioned) statement DELETE statement DELETE (positioned) statement INSERT statement
<b>Description</b>	<p>Inserts a row into the named cursor. Values for the columns are taken from the SQLDA or the host variable list in a one-to-one correspondence with the columns in the INSERT statement (for an INSERT cursor) or the columns in the select list (for a SELECT cursor).</p> <p>If the <b>sqldata</b> pointer in the SQLDA is the null pointer, then no value is specified for that column. If the column has a DEFAULT VALUE associated with it, that will be used, otherwise a NULL value will be used. If no values are specified for any of the columns of one particular table involved in the cursor, then no row will be inserted into that table.</p> <p>The second SQLDA or host variable list contains the results of the PUT statement.</p>



The optional **ARRAY** clause can be used to carry out wide puts, which insert more than one row at a time and which may improve performance. The value **nnn** is the number of rows to be inserted. The **SQLDA** must contain  $nnn * (\text{columns per row})$  variables. The first row is placed in **SQLDA** variables 0 to  $(\text{columns per row})-1$ , and so on.

**One table only**

Values can be specified for columns of one table only. Inserting into two different tables through a cursor is not supported.

**Inserting into a cursor**

When inserting into a cursor, the position of the inserted row is unpredictable. If the cursor involves a temporary table, the inserted record will not show up in the current cursor at all.

**Example**

```
EXEC SQL PUT cur_employee FROM :emp_id, :emp_lname;
```

## READ statement

**Function** To read ISQL statements from a file.

**Syntax** **READ** *filename* [ *parameters* ]

**Usage** ISQL.

**Permissions** None.

**Side effects** None.

**See also** PARAMETERS statement

**Description** The READ statement reads a sequence of ISQL statements from the named file. This file can contain any valid ISQL statement including other READ statements. READ statements can be nested to any depth. To find the command file, ISQL will first search the current directory, then the directories specified in the environment variable SQLPATH, then the directories specified in the environment variable PATH. If the named file has no file extension, ISQL also searches each directory for the same file name with the extension SQL.

Parameters can be listed after the name of the command file. These parameters correspond to the parameters named on the PARAMETERS statement at the beginning of the statement file (see "PARAMETERS statement" on page 1118). ISQL will then substitute the corresponding parameter wherever the source file contains

```
{ parameter-name }
```

where *parameter-name* is the name of the appropriate parameter.

The parameters passed to a command file can be identifiers, numbers, quoted identifiers, or strings. When quotes are used around a parameter, the quotes are put into the text during the substitution. Parameters which are not identifiers, numbers, or strings (contain spaces or tabs) must be enclosed in square brackets ([ ]). This allows for arbitrary textual substitution in the command file.

If not enough parameters are passed to the command file, ISQL prompts for values for the missing parameters.

**Examples** The following are examples of the READ statement.

```
READ status.rpt '160'  
READ birthday.sql [ >= '1988-1-1' ] [ <= '1988-1-30' ]
```

## RELEASE SAVEPOINT statement

<b>Function</b>	Release a savepoint within the current transaction.
<b>Syntax</b>	<b>RELEASE SAVEPOINT</b> [ <i>savepoint-name</i> ]
<b>Usage</b>	Anywhere.
<b>Permissions</b>	There must have been a corresponding SAVEPOINT within the current transaction.
<b>Side effects</b>	None.
<b>See also</b>	SAVEPOINT statement ROLLBACK TO SAVEPOINT statement
<b>Description</b>	<p>Release a savepoint. The <i>savepoint-name</i> is an identifier specified on a SAVEPOINT statement within the current transaction. If <i>savepoint-name</i> is omitted, the most recent savepoint is released.</p> <p>For a description of savepoints, see "Savepoints within transactions" in the chapter "Using Transactions and Locks". Releasing a savepoint does not do any type of commit. It simply removes the savepoint from the list of currently active savepoints.</p>

## RESIGNAL statement

<b>Function</b>	Resignal an exception condition.
<b>Syntax</b>	<b>RESIGNAL</b> [ <i>exception-name</i> ]
<b>Usage</b>	Within an exception handler in a procedure or trigger.
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>See also</b>	SIGNAL statement Compound statements "Using exception handlers in procedures and triggers" in the chapter "Using Procedures, Triggers, and Batches"
<b>Description</b>	Within an exception handler, RESIGNAL allows you to quit the compound statement with the exception still active, or to quit reporting another named exception. The exception will be handled by another exception handler or returned to the application. Any actions by the exception handler before the RESIGNAL are not undone.
<b>Example</b>	The following fragment returns all exceptions except for Column Not Found to the application.

```
...  
DECLARE COLUMN_NOT_FOUND EXCEPTION FOR SQLSTATE  
'52003';  
...  
EXCEPTION  
WHEN COLUMN_NOT_FOUND THEN  
SET message='Column not found' ;  
WHEN OTHERS THEN  
RESIGNAL ;
```

# RESUME statement

<b>Function</b>	To resume a procedure following a query.
<b>Syntax</b>	Syntax 1. <b>RESUME</b> <i>cursor-name</i>  Syntax 2. <b>RESUME</b> [ <b>ALL</b> ]
<b>Parameters</b>	<i>cursor-name</i> : <i>identifier, or host-variable</i>
<b>Usage</b>	Syntax 1: Embedded SQL, procedures, triggers, and batches. Syntax 1 using a <b>host-variable</b> is for Embedded SQL only. Syntax 2: ISQL.
<b>Permissions</b>	The cursor must have been previously opened.
<b>Side effects</b>	None.
<b>See also</b>	DECLARE CURSOR statement "Returning results from procedures" in the chapter "Using Procedures, Triggers, and Batches"
<b>Description</b>	This statement resumes execution of a procedure that returns result sets. The procedure executes until the next result set (SELECT statement with no INTO clause) is encountered. If the procedure completes and no result set is found, the SQLSTATE_PROCEDURE_COMPLETE warning is set. This warning is also set when you RESUME a cursor for a SELECT statement.  The ISQL RESUME statement (Format 2) resumes the current procedure. If ALL is not specified, executing RESUME displays the next result set or, if no more result sets are returned, completes the procedure.  The ISQL RESUME ALL statement cycles through all result sets in a procedure, without displaying them, and completes the procedure. This is useful mainly in testing procedures.
<b>Examples</b>	Embedded SQL example  <ol style="list-style-type: none"> <li>1. EXEC SQL RESUME cur_employee;</li> <li>2. EXEC SQL RESUME :cursor_var;</li> </ol> ISQL examples  <pre>CALL sample_proc() ; RESUME ALL;</pre>

## RETURN statement

<b>Function</b>	To exit from a function or procedure unconditionally, optionally providing a return value. Statements following RETURN are not executed.
<b>Syntax</b>	<b>RETURN</b> [ ( <i>expression</i> ) ]
<b>Usage</b>	Procedures, triggers, and batches
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>See also</b>	CREATE FUNCTION statement CREATE PROCEDURE statement Compound statements
<b>Description</b>	<p>A RETURN statement causes an immediate exit from the function or procedure. If <i>expression</i> is supplied, the value of <i>expression</i> is returned as the value of the function or procedure.</p> <p>Within a function, the expression should be of the same data type as the function's RETURNS data type.</p> <p>RETURN for procedures is for use with Transact-SQL-compatible procedures. For information, see "Transact-SQL Procedure Language".</p>
<b>Example</b>	<p>The following function returns the product of three numbers:</p>

```
CREATE FUNCTION product ( a numeric,  
                          b numeric ,  
                          c numeric)  
  
RETURNS numeric  
BEGIN  
    RETURN ( a * b * c ) ;  
END
```

Calculate the product of three numbers:

```
SELECT product (2, 3, 4)
```

---

**product(2, 3, 4)**

24

The following procedure uses the RETURN statement to avoid executing a complex query if it is meaningless:

```
CREATE PROCEDURE customer_products  
( in customer_id integer DEFAULT NULL)  
RESULT ( id integer, quantity_ordered integer )  
BEGIN
```

```
IF customer_id NOT IN (SELECT id FROM customer)
OR customer_id IS NULL THEN
  RETURN
ELSE
  SELECT product.id, sum(
    sales_order_items.quantity )
  FROM product,
    sales_order_items,
    sales_order
  WHERE sales_order.cust_id=customer_id
  AND sales_order.id=sales_order_items.id
  AND sales_order_items.prod_id=product.id
  GROUP BY product.id
END IF
END
```

# REVOKE statement

**Function** To remove permissions for specified user(s).

**Syntax** Syntax 1

```

REVOKE
    {
        CONNECT
    | DBA
    | RESOURCE
    | GROUP
    | MEMBERSHIP IN GROUP userid,...
    }
... FROM userid,...

```

Syntax 2

```

REVOKE
    {
        ALTER
    | DELETE
    | INSERT
    | REFERENCE ,...
    | SELECT
    | UPDATE [(column-name,...)]
    | ALL [PRIVILEGES]
    }
... ON [owner.]table-name FROM userid,...

```

Syntax 3

```

REVOKE EXECUTE ON [owner.]procedure-name FROM userid,...

```

**Usage** Anywhere.

**Permissions** Must be the grantor of the permissions that are being revoked or must have DBA authority.

If revoking CONNECT permissions or revoking table permissions from another user, the other user must not be connected to the database.

**Side effects** Automatic commit.

**See also** GRANT statement

**Description** The REVOKE statement is used to remove permissions that were given using the GRANT statement. Form 1 is used to revoke special user permissions and Form 2 is used to revoke table permissions. Form 3 is used to revoke permission to execute a procedure. REVOKE CONNECT is used to remove a user ID from a database. REVOKE GROUP will automatically REVOKE MEMBERSHIP from all members of the group.



**Examples**

Prevent user Dave from updating the employee table.

```
REVOKE UPDATE ON employee FROM dave ;
```

Revoke resource permissions from user Jim.

```
REVOKE RESOURCE FROM Jim ;
```

Disallow the Finance group from executing the procedure sp\_customer\_list.

```
REVOKE EXECUTE ON sp_customer_list  
FROM finance ;
```

Drop user ID FranW from the database.

```
REVOKE CONNECT FROM FranW
```

## **REVOKE CONSOLIDATE statement**

<b>Function</b>	To stop a consolidated database from receiving SQL Remote messages from this database.
<b>Syntax</b>	<b>REVOKE CONSOLIDATE FROM</b> <i>userid</i> ,...
<b>Usage</b>	Anywhere. This statement is applicable only to SQL Remote.
<b>Permissions</b>	Must have DBA authority.
<b>Side effects</b>	Automatic commit. Drops all subscriptions for the user.
<b>See also</b>	GRANT CONSOLIDATE statement
<b>Description</b>	CONSOLIDATE permissions must be granted at a remote database for the user ID representing the consolidated database. The REVOKE CONSOLIDATE statement removes the consolidated database user ID from the list of users receiving messages from the current database.
<b>Example</b>	<pre>REVOKE CONSOLIDATE FROM condb</pre>

# REVOKE PUBLISH statement

<b>Function</b>	To terminate the identification of the named user ID as the CURRENT publisher.
<b>Syntax</b>	<b>REVOKE PUBLISH FROM</b> <i>userid</i>
<b>Usage</b>	Anywhere. This statement is applicable only to SQL Remote.
<b>Permissions</b>	Must have DBA authority.
<b>Side effects</b>	Automatic commit.
<b>See also</b>	GRANT PUBLISH statement REVOKE REMOTE statement CREATE PUBLICATION statement CREATE SUBSCRIPTION statement
<b>Description</b>	<p>Each database in a SQL Remote installation is identified in outgoing messages by a <b>publisher</b> user ID. The current publisher user ID can be found using the CURRENT PUBLISHER special constant. The following query identifies the current publisher:</p> <pre>SELECT CURRENT PUBLISHER</pre> <p>The REVOKE PUBLISH statement ends the identification of the named user ID as the publisher.</p> <p>You should not REVOKE PUBLISH from a database while the database has active SQL Remote publications or subscriptions.</p> <p>Issuing a REVOKE PUBLISH statement at a database has several consequences for a SQL Remote installation:</p> <ul style="list-style-type: none"> <li>◆ You will not be able to insert data into any tables with a CURRENT PUBLISHER column as part of the primary key. publisher user ID, and so will not be accepted by recipient databases.</li> </ul> <p>If you change the publisher user ID at any consolidated or remote database in a SQL Remote installation, you must ensure that the new publisher user ID is granted REMOTE permissions on all databases receiving messages from the database. This will generally require all subscriptions to be dropped and recreated.</p>
<b>Example</b>	REVOKE PUBLISH FROM publisher_ID

## **REVOKE REMOTE statement**

<b>Function</b>	To stop a user from being able to receive SQL Remote messages from this database.
<b>Syntax</b>	<b>REVOKE REMOTE FROM</b> <i>userid</i> ,...
<b>Usage</b>	Anywhere. This statement is applicable only to SQL Remote.
<b>Permissions</b>	Must have DBA authority.
<b>Side effects</b>	Automatic commit. Drops all subscriptions for the user.
<b>Description</b>	REMOTE permissions are required for a user ID to receive messages in a SQL Remote replication installation. The REVOKE REMOTE statement removes a user ID from the list of users receiving messages from the current database.
<b>Example</b>	<pre>REVOKE REMOTE FROM SamS</pre>

## ROLLBACK statement

<b>Function</b>	To undo any changes made since the last commit or rollback.
<b>Syntax</b>	<b>ROLLBACK [ WORK ]</b>
<b>Usage</b>	Anywhere.
<b>Permissions</b>	Must be connected to the database.
<b>Side effects</b>	Closes all cursors not opened WITH HOLD.
<b>See also</b>	COMMIT statement ROLLBACK TO SAVEPOINT statement
<b>Description</b>	The ROLLBACK statement ends a logical unit of work (transaction) and undoes all changes made to the database during this transaction. A <b>transaction</b> is the database work done between COMMIT or ROLLBACK statements on one database connection.

## **ROLLBACK TO SAVEPOINT statement**

<b>Function</b>	To cancel any changes made since a SAVEPOINT.
<b>Syntax</b>	<b>ROLLBACK TO SAVEPOINT</b> [ <i>savepoint-name</i> ]
<b>Usage</b>	Anywhere.
<b>Permissions</b>	There must have been a corresponding SAVEPOINT within the current transaction.
<b>Side effects</b>	None.
<b>See also</b>	SAVEPOINT statement RELEASE SAVEPOINT statement ROLLBACK statement
<b>Description</b>	<p>The ROLLBACK TO SAVEPOINT statement will undo any changes that have been made since the SAVEPOINT was established. Changes made prior to the SAVEPOINT are not undone; they are still pending. For a description of savepoints, see "Savepoints within transactions" in the chapter "Using Transactions and Locks".</p> <p>The <i>savepoint-name</i> is an identifier that was specified on a SAVEPOINT statement within the current transaction. If <i>savepoint-name</i> is omitted, the most recent savepoint is used. Any savepoints since the named savepoint are automatically released.</p>

## ROLLBACK TRIGGER statement

<b>Function</b>	To undo any changes made in a trigger.
<b>Syntax</b>	<b>ROLLBACK TRIGGER [ WITH <i>raiserror-statement</i> ]</b>
<b>Usage</b>	In a trigger
<b>Permissions</b>	Must be connected to the database.
<b>Side effects</b>	None
<b>See also</b>	CREATE TRIGGER statement ROLLBACK statement ROLLBACK TO SAVEPOINT statement "Transact-SQL RAISERROR statement" in the chapter "Transact-SQL Procedure Language"
<b>Description</b>	<p>The ROLLBACK TRIGGER statement rolls back the work done in a trigger, including the data modification that caused the trigger to fire.</p> <p>Optionally, a RAISERROR statement can be issued. If a RAISERROR statement is issued, an error is returned to the application. If no RAISERROR statement is issued, no error is returned.</p> <p>If a ROLLBACK TRIGGER statement is used within a nested trigger and without a RAISERROR statement, only the innermost trigger and the statement which caused it to fire are undone.</p>

## SAVEPOINT statement

<b>Function</b>	To establish a savepoint within the current transaction.
<b>Syntax</b>	<b>SAVEPOINT</b> [ <i>savepoint-name</i> ]
<b>Usage</b>	Anywhere.
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>See also</b>	RELEASE SAVEPOINT statement ROLLBACK TO SAVEPOINT statement
<b>Description</b>	<p>Establish a savepoint within the current transaction. The <i>savepoint-name</i> is an identifier that can be used in an RELEASE SAVEPOINT or ROLLBACK TO SAVEPOINT statement. All savepoints are automatically released when a transaction ends. See "Savepoints within transactions" in the chapter "Using Transactions and Locks".</p> <p>Savepoints that are established while a trigger is executing or while an atomic compound statement is executing are automatically released when the atomic operation ends.</p>



# SELECT statement

<b>Function</b>	To retrieve information from the database.
<b>Syntax</b>	<pre> <b>SELECT</b> [ <b>ALL</b>   <b>DISTINCT</b> ] <i>select-list</i>     ...[ <b>INTO</b> { <i>host-variable-list</i>   <i>variable-list</i> } ]     ...[ <b>FROM</b> <i>table-list</i> ]     ...[ <b>WHERE</b> <i>search-condition</i> ]     ...[ <b>GROUP BY</b> <i>column-name</i>, ... ]     ...[ <b>HAVING</b> <i>search-condition</i> ]     ...[ <b>ORDER BY</b> { <i>expression</i>   <i>integer</i> } [ <b>ASC</b>   <b>DESC</b> ], ... ] </pre>
<b>Parameters</b>	<pre> <i>select-list</i>:     <i>table-name</i>       <i>expression</i> [ [ <b>AS</b> ] <i>alias-name</i> ]       * </pre>
<b>Usage</b>	<p>Anywhere</p> <p>The INTO clause with <i>host-variable-list</i> is used in Embedded SQL only.</p> <p>The INTO clause with <i>variable-list</i> is used in procedures and triggers only.</p>
<b>Permissions</b>	Must have SELECT permission on the named tables and views.
<b>Side effects</b>	None.
<b>See also</b>	<p>CREATE VIEW statement</p> <p>DECLARE CURSOR statement</p> <p>"Expressions" in the chapter "Watcom-SQL Language Reference"</p> <p>FETCH statement</p> <p>FROM clause</p> <p>OPEN statement</p> <p>"Search conditions" in the chapter "Watcom-SQL Language Reference"</p> <p>UNION operation</p>
<b>Description</b>	<p>The SELECT statement is used for retrieving results from the database.</p> <p>A SELECT statement can be used in ISQL to browse data in the database or to export data from the database to an external file.</p> <p>A SELECT statement can also be used in procedures and triggers or in Embedded SQL. The SELECT statement with an INTO clause is used for retrieving results from the database when the SELECT statement only returns one row. For multiple row queries, you must use cursors.</p> <p>A SELECT statement can also be used to return a result set from a procedure. The various parts of the SELECT statement are described below:</p>

**ALL or DISTINCT** If neither ALL nor DISTINCT is specified, ALL rows which satisfy the clauses of the SELECT statement are retrieved. If DISTINCT is specified, duplicate output rows are eliminated. This is called the **projection** of the result of the statement. In many cases, statements take significantly longer to execute when DISTINCT is specified. Thus, the use of DISTINCT should be reserved for cases where it is necessary.

If DISTINCT is used, the statement cannot contain an aggregate function with a DISTINCT parameter.

**select list** The select list is a list of expressions separated by commas specifying what will be retrieved from the database. If asterisk (\*) is specified, it is expanded to select all columns of all tables in the FROM clause (**table-name** all columns of the named table). Aggregate functions are allowed in the select list (see "Watcom-SQL Functions"). Subqueries are also allowed in the select list (see "Expressions" in the chapter "Watcom-SQL Language Reference"). Each subquery must be within parentheses.

Alias-names can be used throughout the query to represent the aliased expression.

Alias names are also displayed by ISQL at the top of each column of output from the SELECT statement. If the optional alias name is not specified after an expression, ISQL will display the expression.

**INTO host-variable-list** specifies where the results of the SELECT statement will go. There must be one host-variable item for each item in the select list. Select list items are put into the host variables in order. An indicator host variable is also allowed with each **host-variable** so the program can tell if the select list item was NULL.

**INTO variable-list** This clause is used in procedures and triggers only. It specifies where the results of the SELECT statement will go. There must be one variable for each item in the select list. Select list items are put into the variables in order.

**FROM table-list** Rows are retrieved from the tables and views specified in the **table list**. Joins can be specified using join operators. For more information, see "FROM clause" on page 1074. A SELECT statement with no FROM clause can be used to display the values of expressions not derived from tables. For example:

```
SELECT @@version
```

displays the value of the global variable @@version. This is equivalent to:

```
SELECT @@version  
FROM DUMMY
```

**WHERE search-condition** that will be selected from the tables named in the FROM clause. It is also used to do joins between multiple tables. This is accomplished by putting a condition in the WHERE clause that relates a column or group of columns from one table with a column or group of columns from another table. Both tables must be listed in the FROM clause.

See "Search conditions" in the chapter "Watcom-SQL Language Reference" for a full description.

**GROUP BY { column-name | alias | function }, ...** database. You can group by columns or alias names or functions. GROUP BY expressions must also appear in the select list. The result of the query contains one row for each distinct set of values in the named columns, aliases, or functions. The resulting rows are often referred to as **groups** since there is one row in the result for each group of rows from the table list. For the sake of GROUP BY, all NULL values are treated as identical. Aggregate functions can then be applied to these groups to get meaningful results.

When GROUP BY is used, the select list, HAVING clause and ORDER BY clause cannot reference any identifiers except those named in the GROUP BY clause. The exception is that the select list and HAVING clause may contain aggregate functions.

**HAVING search-condition** based on the group values and not on the individual row values. The HAVING clause can only be used if either the statement has a GROUP BY clause or if the select list consists solely of aggregate functions. Any column names referenced in the HAVING clause must either be in the GROUP BY clause or be used as a parameter to an aggregate function in the HAVING clause.

**ORDER BY expression, ...** results of a query. Each item in the ORDER BY list can be labeled as ASC for ascending order or DESC for descending order. Ascending is assumed if neither is specified. If the expression is an integer N, then the query results will be sorted by the N'th item in the select list.

In Embedded SQL, the SELECT statement is used for retrieving results from the database and placing the values into host variables via the INTO clause. The SELECT statement must return only one row. For multiple row queries, you must use cursors.

### Embedded SQL Example Examples

```
SELECT count(*) INTO :size FROM employee
```

List all the tables and views in the system catalog.

```
SELECT tname
FROM SYS.SYSCATALOG
WHERE tname LIKE 'SYS%' ;
```

List all customers and the total value of their orders.

```
SELECT company_name,  
       CAST( sum(sales_order_items.quantity *  
               product.unit_price) AS INTEGER) VALUE  
FROM customer  
   LEFT OUTER JOIN sales_order  
   LEFT OUTER JOIN sales_order_items  
   LEFT OUTER JOIN product  
GROUP BY company_name  
ORDER BY VALUE DESC
```

How many employees are there?

```
SELECT count(*)  
FROM Employee ;
```

# SET statement

<b>Function</b>	To assign a value to a SQL variable.
<b>Syntax</b>	<b>SET</b> <i>identifier</i> = <i>expression</i>
<b>Usage</b>	Anywhere.
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>See also</b>	CREATE VARIABLE statement DROP VARIABLE statement "Expressions" in the chapter "Watcom-SQL Language Reference"
<b>Description</b>	<p>The SET statement assigns a new value to a variable that was previously created using the CREATE VARIABLE statement.</p> <p>A variable can be used in a SQL statement anywhere a column name is allowed. If there is no column name that matches the identifier, the database engine checks to see if there is a variable that matches and uses its value.</p> <p>Variables are local to the current connection, and disappear when you disconnect from the database or when you use the DROP VARIABLE statement. They are not affected by COMMIT or ROLLBACK statements.</p> <p>Variables are necessary for creating large text or binary objects for INSERT or UPDATE statements from Embedded SQL programs because Embedded SQL host variables are limited to 32,767 bytes.</p>

## Example

- ◆ The following code fragment could be used to insert a large text value into the database.

```
EXEC SQL BEGIN DECLARE SECTION;
char buffer[5001];
EXEC SQL END DECLARE SECTION;

EXEC SQL CREATE VARIABLE hold_text LONG VARCHAR;
EXEC SQL SET hold_text = '';
for(;;) {
    /* read some data into buffer ... */
    size = fread( buffer, 1, 5000, fp );
    if( size <= 0 ) break;

    /* buffer must be null-terminated */
    buffer[size] = '\0';
    /* add data to blob using concatenation */
    EXEC SQL SET hold_text = hold_text || :buffer;
}
EXEC SQL INSERT INTO some_table VALUES ( 1,
hold_text );
```

```
EXEC SQL DROP VARIABLE hold_text;
```

- ◆ The following code fragment could be used to insert a large binary value into the database.

```
EXEC SQL BEGIN DECLARE SECTION;
DECL_BINARY( 5000 ) buffer;
EXEC SQL END DECLARE SECTION;
EXEC SQL CREATE VARIABLE hold_blob LONG BINARY;
EXEC SQL SET hold_blob = '';
for(;;) {
    /* read some data into buffer ... */
    size = fread( &(buffer.array), 1, 5000, fp );
    if( size <= 0 ) break;
    buffer.len = size;

    /* add data to blob using concatenation
       Note that concatenation works for
       binary data too! */
    EXEC SQL SET hold_blob = hold_blob || :buffer;
}
EXEC SQL INSERT INTO some_table VALUES ( 1,
hold_blob );
EXEC SQL DROP VARIABLE hold_blob;
```

# SET CONNECTION statement

<b>Function</b>	To change the active database connection.
<b>Syntax</b>	<b>SET CONNECTION</b> [ <i>connection-name</i> ]
<b>Parameters</b>	<i>connection-name</i> : <i>identifier, string or host-variable</i>
<b>Usage</b>	ISQL.
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>See also</b>	CONNECT statement DISCONNECT statement
<b>Description</b>	The SET CONNECTION statement changes the active database connection to <b>connection-name</b> . The current connection state is saved and will be resumed when it again becomes the active connection. If <b>connection-name</b> is omitted and there is a connection that was not named, that connection becomes the active connection.

### Cursors and connections

When cursors are opened in Embedded SQL, they are associated with the current connection. When the connection is changed, the cursor names will not be accessible. The cursors remain active and in position and will become accessible when the associated connection becomes active again.

**Example** The following example is in Embedded SQL.

```
EXEC SQL SET CONNECTION :conn_name;
```

**Example** From ISQL, set the current connection to the connection named conn1.

```
SET CONNECTION conn1 ;
```

## SET DESCRIPTOR statement

<b>Function</b>	Describes the variables in a SQL descriptor area, and places data into the descriptor area.
<b>Syntax</b>	<b>SET DESCRIPTOR</b> <i>descriptor-name</i> ...{ <b>COUNT</b> = { <i>integer</i>   <i>hostvar</i> }   <b>VALUE</b> <i>n</i> <i>assignment</i> [,...]
<b>Parameters</b>	<i>assignment</i> : { <b>TYPE</b>   <b>SCALE</b>   <b>PRECISION</b>   <b>LENGTH</b>   <b>INDICATOR</b> } ... = { <i>integer</i>   <i>hostvar</i> }   <b>DATA</b> = <i>hostvar</i>
<b>Usage</b>	Embedded SQL.
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>See also</b>	ALLOCATE DESCRIPTOR statement DEALLOCATE DESCRIPTOR statement "The SQL descriptor area (SQLDA)" in the chapter "The Embedded SQL Interface"
<b>Description</b>	<p>The Set descriptor statement is used to describe the variables in a descriptor area, and to place data into the descriptor area.</p> <p>The SET ... COUNT statement sets the number of described variables within the descriptor area. The value for count must not exceed the number of variables specified when the descriptor area was allocated.</p> <p>The value <i>n</i> specifies the variable in the descriptor area upon which the assignment(s) will be performed.</p> <p>Type checking is performed when doing SET ... DATA to ensure that the variable in the descriptor area has the same type as the host variable.</p> <p>If an error occurs, the code is returned in the SQLCA.</p>
<b>Example</b>	For an example, see "ALLOCATE DESCRIPTOR statement" on page 970.



## SET OPTION statement

<b>Function</b>	To change ISQL and database options.
<b>Syntax</b>	<p>Syntax 1. <b>SET [ TEMPORARY ] OPTION</b>  ... [ <i>userid.</i>   <b>PUBLIC.</b> ] <i>option-name</i> = [ <i>option-value</i> ]</p> <p>Syntax 2. <b>SET PERMANENT</b></p> <p>Syntax 3. <b>SET</b></p>
<b>Parameters</b>	<p><i>userid:</i>            <i>identifier, string</i> or <i>host-variable</i></p> <p><i>option-name:</i>    <i>identifier, string</i> or <i>host-variable</i></p> <p><i>option-value:</i>    <i>host-variable</i> (indicator allowed), <i>string, identifier, or number</i></p> <p>Note: Syntax 2 and 3 are for ISQL options only.</p>
<b>Usage</b>	All (Syntax 1), ISQL (all syntaxes).
<b>Permissions</b>	None required to set your own options. Must have DBA authority to set database options for another user or PUBLIC.
<b>Side effects</b>	If TEMPORARY is not specified, an automatic commit is performed.
<b>See also</b>	<p>CONFIGURE statement</p> <p>GET OPTION statement</p> <p>"The database engine" in the chapter "SQL Anywhere Components"</p> <p>The classes of options are:</p> <ul style="list-style-type: none"> <li>◆ General database options</li> <li>◆ Transact-SQL compatibility database options</li> <li>◆ Replication database options</li> <li>◆ ISQL options</li> </ul>
<b>Description</b>	<p>The SET statement is used to change options for both the database and the ISQL environment. The <i>userid</i> parameter can be specified to change someone else's options, however, you must have DBA authority to do so. Specifying TEMPORARY will change the option setting for the current connection only. In this case, <i>userid</i> cannot be specified.</p> <p>Also, in Embedded SQL, only database options can be set temporarily.</p>

Changing a PUBLIC option sets the option value for any userid which has not SET their own value for that option. An option cannot be set for a user ID other than PUBLIC unless there is already a PUBLIC setting for that option. Only a user with DBA authority is allowed to set options for another user ID (including PUBLIC).

If *option-value* is omitted, the specified option setting will be deleted from the database. If it was a personal option setting, then the value used will revert back to the PUBLIC setting. If a TEMPORARY option is deleted, the option setting will revert back to the permanent setting.

**Temporary setting**

Only your own ISQL and database options can be made TEMPORARY; options for other software and any options set for another user produces an error if TEMPORARY is specified.

SET PERMANENT (syntax 2) stores all current ISQL options in the SYS.SYSOPTIONS table in the database. These settings are automatically established every time ISQL is started for the current user ID.

Syntax 3 is used to display all of the current option settings. If there are temporary options set for ISQL or the database engine, these will be displayed; otherwise, the permanent option settings are displayed.

☞ For more information on the different classes of options, see:

- ◆ "Database options", next
- ◆ "
- ◆ "Replication options" on page 1163
- ◆ "ISQL options" on page 1164

## Database options

This section lists all database options with the exception of Transact-SQL compatibility database options and replication database options.

OPTION	VALUES	DEFAULT
BACKGROUND_PRIORITY	ON, OFF	OFF
BLOCKING	ON,OFF	ON
CHECKPOINT_TIME	number of minutes	60
COOPERATIVE_COMMITS	ON,OFF	ON

OPTION	VALUES	DEFAULT
COOPERATIVE_COMMIT_TIMEOUT	integer	250
DATE_FORMAT	string	'YYYY-MM-DD"MM/DD/YYYY' (1)
DATE_ORDER	'YMD','DMY','MDY'	'YMD'MDY' (1)
DELAYED_COMMITS	ON,OFF	OFF
DELAYED_COMMIT_TIMEOUT	integer	500
ISOLATION_LEVEL	0,1,2,3	0
PRECISION	number of digits	30
QUERY_PLAN_ON_OPEN	ON,OFF	OFF
QUOTED_IDENTIFIER	ON,OFF	ON
RECOVERY_TIME	number of minutes	2
ROW_COUNTS	ON,OFF	OFF
SCALE	number of digits	6
SQL_FLAGGER_ERROR_LEVEL	E,I,F,W	W
SQL_FLAGGER_WARNING_LEVEL	E,I,F,W	W
THREAD_COUNT	number of threads	0
TIME_FORMAT	string	'HH:NN:ss.SSS'
TIMESTAMP_FORMAT	string	'YYYY-MM-DD HH:NN:ss.SSS'
WAIT_FOR_COMMIT	ON,OFF	OFF

\$(1) Default for databases created with Watcom SQL 3.0.

#### BACKGROUND\_PRIORITY

When set to ON, requests on the current connection will have minimal impact on the performance of other connections. This option allows tasks for which responsiveness is critical to coexist with other tasks for which performance is not as important. The default setting is OFF.

#### BLOCKING

Controls whether locking conflicts result in one user becoming blocked or an error condition being returned. The default is ON. See "Transaction blocking and deadlock" in the chapter "Using Transactions and Locks".

**CHECKPOINT\_**  
**TIME** Set the maximum desired length of time that the database engine will run without doing a checkpoint. The time is specified in minutes (the default value set by DBINIT is 60). This option is used with the RECOVERY\_TIME option to decide when checkpoints should be done. See "The checkpoint log" in the chapter "Backup and Data Recovery".

Because this is a global option for the database, only the PUBLIC setting is used. Individual settings for users will have no effect. Also, changing this option does not take effect immediately. You must shut down the database engine and restart for the change to take effect.

**COOPERATIVE\_**  
**COMMITTS** If COOPERATIVE\_COMMITTS is set to OFF, a COMMIT is written to disk as soon as it is received by the database engine, and the application is then allowed to continue.

When COOPERATIVE\_COMMITTS is set to ON (the default), the database engine does not immediately write the COMMIT to the disk. Instead, it requires the application to wait for a maximum length set by the COOPERATIVE\_COMMIT\_TIMEOUT option for something else to put on the pages before they are written to disk.

Setting COOPERATIVE\_COMMITTS to ON, and increasing the COOPERATIVE\_COMMIT\_TIMEOUT setting, increases overall database engine throughput by cutting down the number of disk I/Os, but at the expense of a longer turnaround time for each individual connection.

If both COOPERATIVE\_COMMITTS and DELAYED\_COMMITTS are set to ON, then if the COOPERATIVE\_COMMIT\_TIMEOUT interval passes without the pages getting written, the application is resumed (as if the commit had worked), and the remaining interval (DELAYED\_COMMIT\_TIMEOUT - COOPERATIVE\_COMMIT\_TIMEOUT) is used as a DELAYED\_COMMIT interval, after which the pages will be written, even if they are not full.

**COOPERATIVE\_**  
**COMMIT\_**  
**TIMEOUT** This option only has meaning when COOPERATIVE\_COMMITTS is set to ON. The COOPERATIVE\_COMMIT\_TIMEOUT option setting governs when a COMMIT entry in the transaction log is written to disk. With COOPERATIVE\_COMMITTS set to ON, the database engine waits for the number of milliseconds set in the COOPERATIVE\_COMMIT\_TIMEOUT option for other connections to fill a page of the log before writing to disk. The default setting is 250 milliseconds.

**DATE\_FORMAT** Sets the format used for dates retrieved from the database. The format is a string using the following symbols:

Symbol	Description
yy	Two digit year
yyyy	Four digit year
mm	Two digit month, or two digit minutes if following a colon(as in 'hh:mm')
mmm[m...]	Character short form for months—as many characters as there are m's
dd	Two digit day of month
ddd[d...]	Character short form for day of the week
hh	Two digit hours
nn	Two digit minutes
aa	Am or pm (12 hour clock)
pp	PM if needed (12 hour clock)
f	Use French days and months

Each symbol is substituted with the appropriate data for the date being formatted. Any format symbol that represents character rather than digit output can be put in upper case which will cause the substituted characters to also be in upper case. For numbers, using mixed case in the format string will suppress leading zeros.

**Note**

The default date format has changed since Watcom SQL 3.0. The new format corresponds to ISO date format specifications ('YYYY-MM-DD'). Databases created with Watcom SQL 3.0 will still use the old option value. In addition, the new default date format does not specify hours and minutes. `TIMESTAMP_FORMAT` includes hours, minutes and seconds.

**DATE\_ORDER**

The database option `DATE_ORDER` is used to determine whether 10/11/12 is Oct 11 1912, Nov 12 1910, or Nov 10 1912. The option can have the value 'MDY', 'YMD', or 'DMY'.

**Note**

The default date order has changed since Watcom SQL 3.0. The new order ('YMD') corresponds to ISO date format specifications. Databases created with Watcom SQL 3.0 will still use the old option value which was 'MDY'.

**DELAYED\_  
COMMITTS**

When set to ON, the database engine or server replies to a COMMIT statement immediately instead of waiting until the transaction log entry for the COMMIT has been written to disk. When set to OFF, the application must wait until the COMMIT is written to disk. This option must be set to OFF for ANSI/ISO COMMIT behavior.

When this option is ON, the log is written to disk when the log page is full or according to the DELAYED\_COMMIT\_TIMEOUT option setting, whichever is first. There is a slight chance that a transaction may be lost even though committed if a system failure occurs after the engine replies to a COMMIT, but before the page is written to disk. Setting DELAYED\_COMMITTS to ON, and the DELAYED\_COMMIT\_TIMEOUT option to a high value, promotes a quick response time at the cost of security.

If both COOPERATIVE\_COMMITTS and DELAYED\_COMMITTS are set to ON, then if the COOPERATIVE\_COMMIT\_TIMEOUT interval passes without the pages getting written, the application is resumed (as if the commit had worked), and the remaining interval (DELAYED\_COMMIT\_TIMEOUT - COOPERATIVE\_COMMIT\_TIMEOUT) is used as a DELAYED\_COMMIT interval, after which the pages will be written, even if they are not full.

**DELAYED\_  
COMMIT\_  
TIMEOUT**

This option only has meaning when DELAYED\_COMMITTS is set to ON. With DELAYED\_COMMITTS set ON, the DELAYED\_COMMIT\_TIMEOUT option setting governs when a COMMIT entry in the transaction log is written to disk. With DELAYED\_COMMITTS set to ON, the database engine waits for the number of milliseconds set in the DELAYED\_COMMIT\_TIMEOUT option for other connections to fill a page of the log before writing the current page contents to disk. The default setting is 500 milliseconds.

**ISOLATION\_  
LEVEL**

Controls the locking isolation level as follows.

- ◆ **0** Allow dirty reads, nonrepeatable reads, and phantom rows.
- ◆ **1** Prevent dirty reads. Allow nonrepeatable reads and phantom rows.
- ◆ **2** Prevent dirty reads and guarantee repeatable reads. Allow phantom rows.
- ◆ **3** Serializable. Do not allow dirty reads, guarantee repeatable reads, and do not allow phantom rows.

The default is 0. See "Isolation levels and consistency" in the chapter "Using Transactions and Locks".

**PRECISION**

Specifies the maximum number of digits in the result of any decimal arithmetic. Precision is the total number of digits to the left and right of the decimal point. The SCALE option specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum PRECISION.

Multiplication, division, addition, subtraction, and aggregate functions can all have results that exceed the maximum precision.

For example, when a DECIMAL(8,2) is multiplied with a DECIMAL(9,2), the result could require a DECIMAL(17,4). If PRECISION is 15, only 15 digits will be kept in the result. If SCALE is 4, the result will be a DECIMAL(15,4). If SCALE is 2, the result will be a DECIMAL(15,2). In both cases, there is a possibility for overflow.

**RECOVERY\_  
TIME**

Sets the maximum length of time that the database engine will take to recover from system failure. The time is specified in minutes (the default value set by DBINIT is 2). This option is used with the CHECKPOINT\_TIME option to decide when checkpoints should be done. See "Recovery from system failure" in the chapter "Backup and Data Recovery". SQL Anywhere uses a heuristic to measure the recovery time based on the operations since the last checkpoint. Thus, the recovery time is not exact. Because this is a global option for the database, only the PUBLIC setting is used. Individual settings for users will have no effect. Also, changing this option does not take effect immediately. You must shut down the database engine and restart it for the change to take effect.

**ROW\_COUNTS**

Specifies whether the database will always count the number of rows in a query when it is opened. If this option is off, the row count, the row count is usually only an estimate. If this option is on, the row count is always accurate but opening queries may take significantly longer.

**SCALE**

Specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum PRECISION.

Multiplication, division, addition, subtraction, and aggregate functions can all have results that exceed the maximum precision. See the PRECISION option for an example.

**THREAD\_  
COUNT**

Sets the number of execution threads that will be used in the database engine while running with multiple users. This number controls the number of concurrent requests that the database engine will process. A value of 0 (the default) means an operating system specific value, which is 8 for DBENG50S and 20 for all other engines and servers.

Because this is a global option for the database, only the PUBLIC setting is used. Individual settings for users will have no effect. Also, changing this option does not take effect immediately. You must shut down the database engine and restart for the change to take effect.

**TIME\_FORMAT**

Sets the format used for times retrieved from the database. The format is a string using the following symbols:

- ◆ **hh** Two digit hours (24 hour clock)
- ◆ **nn** Two digit minutes
- ◆ **mm** Two digit minutes if following a colon (as in 'hh:mm')
- ◆ **ss[.s...]** Two digit seconds plus optional fraction

Each symbol will be substituted with the appropriate data for the date being formatted. Any format symbol that represents character rather than digit output can be put in uppercase which will cause the substituted characters to also be in uppercase. For numbers, using mixed case in the format string will suppress leading zeros.

**TIMESTAMP\_FORMAT**

Sets the format used for timestamps retrieved from the database. The format is a string using the following symbols:

<b>Symbol</b>	<b>Description</b>
yy	Two digit year
yyyy	Four digit year
mm	Two digit month, or two digit minutes if following a colon(as in 'hh:mm')
mmm[m...]	Character short form for months—as many characters as there are m's
dd	Two digit day of month
ddd[d...]	Character short form for day of the week
hh	Two digit hours
nn	Two digit minutes
ss.ssssss	Seconds and fractions of a second, up to six decimal places. Not all platforms support timestamps to a precision of six places.
aa	Am or pm (12 hour clock)
pp	Pm if needed (12 hour clock)
f	Use French days and months



Each symbol is substituted with the appropriate data for the date being formatted. Any format symbol that represents character rather than digit output can be put in uppercase which will cause the substituted characters to also be in uppercase. For numbers, using mixed case in the format string will suppress leading zeros.

## WAIT\_FOR\_COMMIT

If this option is on, the database will not check foreign key integrity until the next COMMIT statement. Otherwise, all foreign keys not created with the CHECK ON COMMIT option are checked as they are inserted, updated or deleted.

## Compatibility options

Compatibility options are database options included to allow SQL Anywhere behavior to be made compatible with SQL Server, or to support both old behavior and allow ISO SQL/92 behavior.

OPTION	VALUES	DEFAULT
ALLOW_NULLS_BY_DEFAULT	ON, OFF	ON
ANSI_BLANKS	ON, OFF	OFF
ANSI_INTEGER_OVERFLOW	ON, OFF	ON
ANSINULL	ON, OFF	ON
ANSI_PERMISSIONS	ON, OFF	ON
AUTOMATIC_TIMESTAMP	ON, OFF	OFF
CHAINED	ON, OFF	ON
CLOSE_ON_ENDTRANS	ON, OFF	OFF
CONVERSION_ERROR	ON, OFF	ON
DIVISION_BY_ZERO_ERROR	ON, OFF	OFF
FIRE_TRIGGERS	ON, OFF	ON
NEAREST_CENTURY	integer	0
NON_KEYWORDS	comma-separated keywords list	No keywords turned off
QUERY_PLAN_ON_OPEN	ON, OFF	OFF
QUOTED_IDENTIFIER	ON, OFF	ON

<b>OPTION</b>	<b>VALUES</b>	<b>DEFAULT</b>
RI_TRIGGER_TIME	BEFORE, AFTER	AFTER
SQL_FLAGGER_ ERROR_LEVEL	E, I, F, W	W
SQL_FLAGGER_ WARNING_LEVEL	E, I, F, W	W
STRING_ RTRUNCATION	ON, OFF	OFF
TSQL_HEX_CONSTANT	ON, OFF	OFF
TSQL_VARIABLES	ON, OFF	OFF

**ALLOW\_NULLS\_  
BY\_DEFAULT**

Controls whether new columns created without specifying either NULL or NOT NULL are allowed contain NULL values. The ALLOW\_NULLS\_BY\_DEFAULT option is included for Transact-SQL compatibility. The default is ON. See "Setting options for Transact-SQL compatibility" in the chapter "Using Transact-SQL with SQL Anywhere".

**ANSI\_BLANKS**

The ANSI\_BLANKS option has no effect unless the database was created with the -b command-line option. For databases created with the -b command-line option, turning ANSI\_BLANKS on forces a truncation error whenever a value of data type CHAR(N) is read into a C char(M) variable for values of N greater than or equal to M. With ANSI\_BLANKS set to OFF, a truncation error occurs only when the truncated characters include non-blank characters.

If ANSI\_BLANKS is ON, then when supplying a value of data type DT\_STRING, the :name.sqllen:ename. field must be set to the length of the value, including space for the terminating null character. With ANSI\_BLANKS off, the length is determined solely by the position of the null character.

**ANSI\_INTEGER\_  
OVERFLOW**

The ISO SQL/92 standard requires that an arithmetic operation resulting in an integer overflow should result in an error SQLSTATE = 22003 - overflow error. SQL Anywhere behavior was different from this previously. The ANSI\_INTEGER\_OVERFLOW option can be set to OFF to maintain compatibility with previous releases of the software. The default setting is OFF.

**ANSINULL**

With ANSINULL ON, results of comparisons with NULL using '=' or '!=' are unknown. Also, aggregate functions on columns containing NULL values cause the warning 'null value eliminated in aggregate function' (SQLSTATE=01003). ANSINULL 'off' allows comparisons with NULL to yield results that are not unknown, for compatibility with SQL Server, and turns off the warning. The default setting is ON.

**ANSI\_ PERMISSIONS**

With ANSI\_PERMISSIONS ON (the default), SQL92 permissions requirements for delete and update statements are checked. The default value is off in SQL Server. The following table outlines the differences.

SQL Statement	Permissions Required with ansi_permissions off	Permissions Required with ansi_permissions on
UPDATE	update permission on columns where values are being set	update permission on columns where values are being set  select permission on all columns appearing in where clause.  SElect permission on all columns on right side of set clause.
DELETE	delete permission on table	delete permission on table.  select permission on all columns appearing in where clause

The ANSI\_PERMISSIONS option can be set only for the PUBLIC group. No private settings are allowed.

**AUTOMATIC\_ TIMESTAMP**

Controls whether any new columns with the TIMESTAMP data type that do not have an explicit default value defined are given a default value of the Transact-SQL timestamp value. The AUTOMATIC\_TIMESTAMP option is included for Transact-SQL compatibility. The default is OFF. See "Setting options for Transact-SQL compatibility" in the chapter "Using Transact-SQL with SQL Anywhere".

CHAINED	Controls the Transact-SQL transaction mode. In UNCHAINED mode (CHAINED = OFF) each statement is committed individually unless an explicit BEGIN TRANSACTION statement is executed to start a transaction. In chained mode (CHAINED = ON) a transaction is implicitly started before any data retrieval or modification statement. For SQL Anywhere the default setting is ON; for SQL Server the default setting is OFF.
CLOSE_ON_ENDTRANS	When CLOSE_ON_ENDTRANS is set to ON, cursors are closed at the end of a transaction; when set to OFF cursors are not closed. This behavior can be overridden by opening a cursor 'WITH HOLD'. The default is OFF, and ON provides Transact-SQL compatible behavior.
CONVERSION_ERROR	Controls whether conversion errors will be reported by the database, or ignored. If conversion errors are ignored, the NULL value is used in place of the value that could not be converted.
DIVISION_BY_ZERO_ERROR	The DIVIDE_BY_ZERO_ERROR option indicates whether division by zero is reported as an error. If the option is set OFF division by zero is not an error - this is the previous behavior. If the option is set ON, then division by zero results in an error with SQLSTATE 22012. The default setting is ON.
FIRE_TRIGGERS	<p>When set to ON (the default), triggers are fired. When set to OFF, no triggers are fired, including referential integrity triggers (such as cascading updates and deletes). Only a user with DBA authority can set this option. The option is overridden by the -gf command-line option, which turns off all trigger firing regardless of the FIRE_TRIGGERS setting.</p> <p>This option is relevant when replicating data from SQL Server to SQL Anywhere, as all actions from SQL Server transaction logs are replicated to SQL Anywhere, including actions carried out by triggers.</p>
NEAREST_CENTURY	<p>Controls handling of two-digit years, when converting from strings to dates or timestamps. The historical SQL Anywhere behavior is to add 1900 to the year. SQL Server behavior is to use the nearest century, so that if the year value yy is less than 50, then the year is set to 20yy.</p> <p>The NEAREST_CENTURY setting is a numeric value that acts as a roll-over point. Two digit years less than the value are converted to 20yy, while years greater than or equal to the value are converted to 19yy.</p> <p>The default setting is 0, which gives the historical SQL Anywhere behavior.</p>

**NON\_  
KEYWORDS**

This option turns off individual keywords or all keywords introduced since a specific release of the product. This provides a way of ensuring that applications created with older versions of the product are not broken by new keywords. If you have an identifier in your database that is now a keyword, you can either add double quotes around the identifier in all applications or scripts, or you can turn off the keyword using the **NON\_KEYWORDS** option.

In addition to specifying individual keywords, you can turn off all keywords since a specified release using one of the following special values in the list of keywords:

```
keywords_4_0_d, keywords_4_0_c, keywords_4_0_b,
keywords_4_0_a, keywords_4_0, keywords_5_0_01,
keywords_5_0
```

The following statement prevents **TRUNCATE** and **SYNCHRONIZE** from being recognized as keywords:

```
SET OPTION NON_KEYWORDS = 'TRUNCATE, SYNCHRONIZE'
```

The following statement prevents all keywords introduced since release 4.0d from being recognized as keywords:

```
SET OPTION NON_KEYWORDS = 'keywords_4_0_d'
```

Each new setting of this option replaces the previous setting. The following statement clears all previous settings.

```
SET OPTION NON_KEYWORDS =
```

A side effect of the options is that SQL statements using a turned off keyword cannot be used: they produce a syntax error.

**QUERY\_PLAN\_  
ON\_OPEN**

In previous versions of the product, each time an **OPEN** was done on a cursor, the engine would return in the **SQLCA sqlerrmc** field a string representing the query plan (limited to 70 bytes). A more complete description can be obtained using the **EXPLAIN** statement or the **PLAN** function. For this reason, computing and returning the query plan on an **OPEN** is needed only for compatibility with old applications. The **QUERY\_PLAN\_ON\_OPEN** option controls whether the plan is returned on an **OPEN**. By default, the setting is **OFF**.

**QUOTED\_  
IDENTIFIER**

Controls whether strings enclosed in double quotes are interpreted as identifiers (**ON**) or as literal strings (**OFF**). The **QUOTED\_IDENTIFIER** option is included for Transact-SQL compatibility. The default is **ON**. See "Setting options for Transact-SQL compatibility" in the chapter "Using Transact-SQL with SQL Anywhere".

RI\_TRIGGER\_  
TIME

RI\_TRIGGER\_ACTION controls the relative timing of referential integrity checks and trigger actions. The option can be set to either BEFORE or AFTER. When set to AFTER, RI actions are executed AFTER the update or delete. The default setting is AFTER and only the PUBLIC setting can be used: any other setting is ignored.

Prior to 5.5, the referential integrity triggers were always fired BEFORE. Starting with Release 5.5, the RI\_TRIGGER\_TIME option allows RI triggers to be fired before or after, with the default being to fire AFTER.

The reason for the change of default behavior is that BEFORE triggers give possibilities for infinitely recursing triggers when deleting from self-referencing rows.

SQL\_FLAGGER\_  
ERROR\_LEVEL

This option flags any SQL that is not part of a specified set of SQL/92 as an error.

The allowed values of *level* and their meanings are as follows:

- ◆ **E** Flag syntax that is not entry-level SQL/92 syntax
- ◆ **I** Flag syntax that is not intermediate-level SQL/92 syntax
- ◆ **F** Flag syntax that is not full-SQL/92 syntax
- ◆ **W** Allow all supported syntax

SQL\_FLAGGER\_  
WARNING\_LEVEL

This option flags any SQL that is not part of a specified set of SQL/92 as a warning.

The allowed values of *level* and their meanings are as follows:

- ◆ **E** Flag syntax that is not entry-level SQL/92 syntax
- ◆ **I** Flag syntax that is not intermediate-level SQL/92 syntax
- ◆ **F** Flag syntax that is not full-SQL/92 syntax
- ◆ **W** Allow all supported syntax

STRING\_  
RTRUNCATION

Determines whether an error is raised when an insert or update truncates a char or varchar string. If the truncated characters consist only of spaces, no exception is raised. The setting of ON corresponds to ANSI/ISO SQL/92 behavior. When set to OFF, the exception is not raised and the character string is silently truncated.

TSQL\_HEX\_  
CONSTANT

When set to ON, hexadecimal constants are treated as binary typed constants. To get the historical behavior, set the option to OFF. The default setting is ON.

**TSQL\_  
VARIABLES**

When set to ON, you can use the @ sign instead of the colon as a prefix for host variable names in Embedded SQL. This is implemented primarily for the Open Server Gateway.

**Replication options**

Replication options are database options included to provide control over replication behavior. Some replication options are useful for SQL Remote replication, and some are relevant for use with Sybase Replication Server.

OPTION	VALUES	DEFAULT
DELETE_OLD_LOGS	ON,OFF	OFF
REPLICATE_ALL	ON,OFF	OFF
REPLICATION_ERROR	procedure-name	(no procedure)
VERIFY_THRESHOLD	integer	1000
VERIFY_ALL_COLUMNS	ON,OFF	OFF

**DELETE\_OLD\_  
LOGS**

This option is used by SQL Remote and by the SQL Anywhere Replication Agent. The default setting is OFF. When set to ON, the Message Agent (DBREMOTE) deletes each old transaction log when all the changes it contains have been sent and confirmed as received.

**REPLICATE\_ALL**

This option is used by the SQL Anywhere LTM only. The default setting is OFF. When it is set to ON, the entire database is set to act as a primary site in a Replication Server installation. All changes to the database are sent to Replication Server by the LTM.

**REPLICATION\_  
ERROR**

For SQL Remote, the REPLICATION\_ERROR option allows you to specify a stored procedure called by the Message Agent when a SQL error occurs. By default no procedure is called.

The procedure must have a single argument of type CHAR, VARCHAR, or LONG VARCHAR. The procedure is called once with the SQL error message and once with the SQL statement that causes the error.

While the option allows you to track and monitor SQL errors in replication, you must still design them out of your setup: this option is not intended to resolve such errors.

**VERIFY\_THRESH  
OLD**

This option is used by SQL Remote only. The default setting is 1000. If the data type of a column is longer than the threshold, old values for the column are not verified when an UPDATE is replicated. This keeps the size of SQL Remote messages down, but has the disadvantage that conflicting updates of long values are not detected.

**VERIFY\_ALL\_  
COLUMNS**

This option is used by SQL Remote only. The default setting is OFF. When set to ON, messages containing updates published by the local database are sent with all column values included, and a conflict in any column triggers a RESOLVE UPDATE trigger at the subscriber database.

**ISQL options**

<b>OPTION</b>	<b>VALUES</b>	<b>DEFAULT</b>
AUTO_COMMIT	ON,OFF	OFF
AUTO_REFETCH	ON,OFF	ON
BELL	ON,OFF	ON
COMMAND_DELIMITER	string	';
COMMIT_ON_EXIT	ON,OFF	ON
ECHO	ON,OFF	ON
HEADINGS	ON,OFF	ON
INPUT_FORMAT	ASCII FIXED DIF DBASE DBASEII DBASEIII FOXPRO LOTUS WATFILE	ASCII
ISQL_LOG	file-name	"
NULLS	string	'(NULL)'
ON_ERROR	STOP CONTINUE PROMPT EXIT NOTIFY_CONTINUE NOTIFY_STOP NOTIFY_EXIT	PROMPT
OUTPUT_FORMAT	TEXT ASCII FIXED DIF DBASEII DBASEIII FOXPRO LOTUS SQL WATFILE	ASCII
OUTPUT_LENGTH	integer	0



OPTION	VALUES	DEFAULT
STATISTICS	0,3,4,5,6	3
TRUNCATION_LENGTH	integer	30

**AUTO\_COMMIT**

If **AUTO\_COMMIT** is *on*, a database COMMIT is performed after each successful statement and a ROLLBACK after each failed statement. Otherwise, a COMMIT or ROLLBACK is only performed when the user issues a COMMIT or ROLLBACK statement or if the user issues a SQL statement which causes an automatic commit such as the CREATE TABLE statement.

**AUTO\_REFETCH**

If **AUTO\_REFETCH** is *on*, then the current query results displayed in the Data window will be refetched from the database after **any** INSERT, UPDATE or DELETE statement. Depending on how complicated the query is, this may take some time. For this reason, it can be turned *off*.

**BELL**

Controls whether the bell will sound when an error occurs.

**COMMAND\_DELIMITER**

Sets the string indicating the termination of a statement in ISQL. The default setting is a single semicolon ';'. This must be changed in order to create triggers and procedures from ISQL using the CREATE TRIGGER or CREATE PROCEDURE statement. The change prevents the semi-colons terminating individual statements within the trigger or procedure from being interpreted as the termination of the CREATE statement itself.

A common setting for this option is '\'.

If the command delimiter is set to a string beginning with a character that is valid in identifiers, the command delimiter must be preceded by a space.

**COMMIT\_ON\_EXIT**

Controls which of a COMMIT or ROLLBACK is done when leaving ISQL. With **COMMIT\_ON\_EXIT** is set to ON (the default), a COMMIT is done; otherwise a ROLLBACK is done.

**ECHO**

Controls whether statements are echoed before they are executed. This is most useful when using the READ statement to execute an ISQL command file.

**HEADINGS**

Controls whether headings will be displayed for the results of a SELECT statement.

**INPUT\_FORMAT**

Sets the default data format expected by the INPUT statement.

Certain file formats contain information about column names and types. Using this information, the INPUT statement will create the database table if it does not already exist. This is a very easy way to load data into the database. The formats that have enough information to create the table are: DBASEII, DBASEIII, DIF, FOXPRO, LOTUS, and WATFILE.

Allowable input formats are:

**ASCII** Input lines are assumed to be ASCII characters, one row per line, with values separated by commas. Alphabetic strings may be enclosed in apostrophes (single quotes) or quotation marks (double quotes). Strings containing commas must be enclosed in either single or double quotes. If single or double quotes are used, double the quote character to use it within the string. Optionally, you can use the DELIMITED BY clause to specify a different delimiter string than the default which is a comma.

Three other special sequences are also recognized. The two characters \n represent a newline character, \\ represents a single , and the sequence \xDD represents the character with hexadecimal code DD.

**DBASE** The file is in dBASE II or dBASE III format. ISQL will attempt to determine which of the two DBase formats the file is based on information in the file. If the table doesn't exist, it will be created.

**DBASEII** The file is in dBASE II format. If the table doesn't exist, it will be created.

**DBASEIII** The file is in dBASE III format. If the table doesn't exist, it will be created.

**DIF** Input file is in Data Interchange Format. If the table doesn't exist, it will be created.

**FIXED** Input lines are in fixed format. The width of the columns can be specified using the COLUMN WIDTHS clause. If they are not specified, then column widths in the file must be the same as the maximum number of characters required by any value of the corresponding database column's type.

**FOXPRO** The file is in FoxPro format (the FoxPro memo field is different than the dBASE memo field). If the table doesn't exist, it will be created.

**LOTUS** The file is a Lotus WKS format worksheet. INPUT assumes that the first row in the Lotus WKS format worksheet is column names. If the table doesn't exist, it will be created. In this case, the types and sizes of the columns created may not be correct because the information in the file pertains to a cell, not to a column.

**WATFILE** The input will be a WATFILE file. If the table doesn't exist, it will be created.

**ISQL\_LOG** If ISQL\_LOG is set to a non-empty string, all ISQL statements are added to the end of the named file. Otherwise, if ISQL\_LOG is set to the empty string ISQL statements are not logged.

**Individual session only**

This is logging of an individual ISQL session only. See "Backup and Data Recovery" for a description of the transaction log which logs all changes to the database by all users.

**NULLS** Specifies how NULL values in the database will be displayed. The default is (NULL) (including the parentheses).

**ON\_ERROR** Controls what happens if an error is encountered while reading statements from a command file as follows:

**STOP** ISQL will stop reading statements from the file and return to the statement window for input.

**PROMPT** ISQL will prompt the user to see if the user wishes to continue.

**CONTINUE** The error will be ignored and ISQL will continue reading statements from the command file. The INPUT statement will continue with the next row, skipping the row that caused the error.

**EXIT** ISQL will terminate.

**NOTIFY\_CONTINUE** The error is displayed in a message box with a single Continue button. Execution continues once the button is clicked.

**NOTIFY\_STOP** The error is displayed in a message box with a single Stop button. Execution of the script stops once the button is clicked.

**NOTIFY\_EXIT** The error is displayed in a message box with a single Exit button. ISQL terminates once the button is clicked.

**OUTPUT\_FORMAT** Sets the output format for the data retrieved by the SELECT statement and redirected into a file. This is also the default output format for the OUTPUT statement. The valid output formats are:

**ASCII** The output is an ASCII format file with one row per line in the file. All values are separated by commas and strings are enclosed in apostrophes (single quotes). The delimiter and quote strings can be changed using the DELIMITED BY and QUOTE clauses. If ALL is specified in the QUOTE clause, then all values (not just strings) will be quoted.

Three other special sequences are also used. The two characters `\n` represent a newline character, `\\` represents a single `,` and the sequence `\xDD` represents the character with hexadecimal code `DD`. This is the default output format.

**DBASEII** The output is a dBASE II format file with the column definitions at the top of the file. Note that a maximum of 32 columns can be output. Also, note that columns longer than 255 characters will be truncated in the file.

**DBASEIII** The output is a dBASE III format file with the column definitions at the top of the file. Note that a maximum of 128 columns can be output. Also, note that columns longer than 255 characters will be truncated in the file.

**DIF** The output is a file in the standard Data Interchange Format.

**FIXED** The output is fixed format with each column having a fixed width. The width for each column can be specified using the `COLUMN WIDTH` clause. If this clause is omitted, the width for each column is computed from the data type for the column, and is large enough to hold any value of that data type. No column headings are output in this format.

**FOXPRO** The output is a FoxPro format file (the FoxPro memo field is different than the dBASE memo field) with the column definitions at the top of the file. Note that a maximum of 128 columns can be output. Also, note that columns longer than 255 characters will be truncated in the file.

**LOTUS** The output is a Lotus WKS format worksheet. Column names will be put as the first row in the worksheet. Note that there are certain restrictions on the maximum size of Lotus WKS format worksheets that other software (such as Lotus 1-2-3) can load. There is no limit to the size of file ISQL can produce.

**SQL** The output is an ISQL INPUT statement required to recreate the information in the table.

**TEXT** The output is a TEXT format file which prints the results in columns with the column names at the top and vertical lines separating the columns. This format is similar to that used to display data in the ISQL data window.

**WATFILE** The output is a WATFILE format file with the column definitions at the top of the file. WATFILE is a tabular file management tool available from Watcom.

---

<b>OUTPUT_LENGTH</b>	Controls the length used when ISQL exports information to an external file (using output redirection or the OUTPUT statement). The default for Output_length is 0—no truncation.
<b>STATISTICS</b>	Controls whether execution times, optimization strategies and other statistics will be displayed in the statistics window. This option can be set to 0, 3, 4, 5, or 6. When 0, the statistics window is not displayed. Otherwise, the value represents the height of the statistics window in lines.
<b>TRUNCATION_LENGTH</b>	When SELECT statement results are displayed on the screen, each column of output is limited to the width of the screen. The TRUNCATION_LENGTH option is used to reduce the width of wide columns so that more than one column will fit on the screen. A value of 0 means that columns will not be truncated.

---

**Examples**

Set the date format option.

```
SET OPTION public.date_format = 'Mmm dd yyyy' ;
```

Set the 'wait for commit' option to on.

```
SET OPTION waitfor_commit = 'on' ;
```

**Embedded SQL Examples**

1. EXEC SQL SET OPTION :user.:option\_name = :value;
2. EXEC SQL SET TEMPORARY OPTION Date\_format = 'mm/dd/yyyy' ;

## SET SQLCA statement

<b>Function</b>	To tell the SQL preprocessor to use a SQLCA other than the default global <i>sqlca</i> .
<b>Syntax</b>	<b>SET SQLCA</b> <i>sqlca</i>
<b>Parameters</b>	<i>sqlca</i> : <i>identifier</i> or <i>string</i>
<b>Usage</b>	Embedded SQL.
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>See also</b>	"Multi-Threaded or Reentrant Code" in the chapter "The Embedded SQL Interface"
<b>Description</b>	The SET SQLCA statement tells the SQL preprocessor to use a SQLCA other than the default global <i>sqlca</i> . The <b>sqlca</b> must be an identifier or string that is a C language reference to a SQLCA pointer. The current SQLCA pointer is implicitly passed to the database interface library on every Embedded SQL statement. All Embedded SQL statements that follow this statement in the C source file will use the new SQLCA. This statement is only necessary when you are writing code that is reentrant (see "Multi-Threaded or Reentrant Code" in the chapter "The Embedded SQL Interface"). The <b>sqlca</b> should reference a local variable. Any global or module static variable is subject to being modified by another thread.
<b>Example</b>	The following function could be found in a Windows DLL. Each application that uses the DLL has its own SQLCA.

```
an_sql_code FAR PASCAL ExecutesSQL( an_application
*app, char *com )
{
EXEC SQL BEGIN DECLARE SECTION;
char *sqlcommand;
EXEC SQL END DECLARE SECTION;
EXEC SQL SET SQLCA "&app->.sqlca";
sqlcommand = com;
EXEC SQL WHENEVER SQLERROR CONTINUE;
EXEC SQL EXECUTE IMMEDIATE :sqlcommand;
return( SQLCODE );
}
```

---

## SIGNAL statement

<b>Function</b>	Signal an exception condition.
<b>Syntax</b>	<b>SIGNAL</b> <i>exception-name</i>
<b>Usage</b>	Procedures, triggers, and batches.
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>See also</b>	RESIGNAL statement Compound statements "Using exception handlers in procedures and triggers" in the chapter "Using Procedures, Triggers, and Batches"
<b>Description</b>	SIGNAL allows you to raise an exception. See "Using exception handlers in procedures and triggers" in the chapter "Using Procedures, Triggers, and Batches" for a description of how exceptions are handled.

# START DATABASE statement

**Function** To start a database on the specified database engine

**Syntax** **START DATABASE** *database-file*  
... [ **AS** *database-name* ]  
... [ **ON** *engine-name* ]  
... [ **AUTOSTOP** { **YES** | **NO** ]

**Usage** ISQL

**Permissions** None

**Side effects** None

**See also** STOP DATABASE statement  
CONNECT statement

**Description** The START DATABASE statement starts a specified database on a specified database engine. The database engine must be running. The full path must be specified for the database-file unless the file is located in the current directory.

The START DATABASE statement does not connect ISQL to the specified database: a CONNECT statement needs to be issued in order to make a connection.

If *database-name* is not specified, a default name is assigned to the database. This default name is the root of the database file. For example, a database in file C:\SQLANY50\SADEMO.DB would be given the default name of sademo.

If *engine-name* is not specified, the default database engine is assumed. The default database engine is the first started engine among those currently running.

The default setting for the AUTOSTOP clause is YES. With AUTOSTOP set to YES, the database is unloaded when the last connection to it is dropped. If AUTOSTOP is set to NO, the database is not unloaded.

**Example** Start the database file C:\SQLANY50\SAMPLE\_2.DB on the current engine.

```
start database 'c:\sqlany50\sample_2.db'
```

Start the database file C:\SQLANY50\SAMPLE\_2.DB as sam2 on the engine named sample.

```
START DATABASE 'c:\sqlany50\sample_2.db'  
AS sam2  
ON sample ;
```



## START ENGINE statement

<b>Function</b>	To start a database engine
<b>Syntax</b>	<b>START ENGINE AS</b> <i>engine-name</i> [ <b>STARTLINE</b> <i>command-string</i> ]
<b>Usage</b>	ISQL
<b>Permissions</b>	None
<b>Side effects</b>	None
<b>See also</b>	STOP ENGINE statement "The database engine" in the chapter "SQL Anywhere Components"
<b>Description</b>	The START ENGINE statement starts a database engine. If you wish to specify a set of options for the engine, use the STARTLINE keyword together with a command string. Valid command strings are those that conform to the database engine command-line description in the chapter "SQL Anywhere Components".
<b>Example</b>	<p>Start a database engine, named sample, without starting any databases on it.</p> <pre>START ENGINE AS sample ;</pre> <p>Start a database engine with a maximum cache size of 4 megabytes, loading the sample database.</p> <pre>START ENGINE AS sample STARTLINE 'dbeng50w -c 4096 c:\sqlany50\sademo.db'</pre> <p>The following example shows the use of a STARTLINE clause.</p> <pre>START ENGINE AS eng1 STARTLINE 'dbeng50 -c 8096'</pre>

## START SUBSCRIPTION statement

<b>Function</b>	To start a subscription for a user to a publication.
<b>Syntax</b>	<b>START SUBSCRIPTION TO</b> <i>publication-name</i> [ ( <i>string</i> ) ] ... <b>FOR</b> <i>userid</i> ,...
<b>Usage</b>	Anywhere. This statement is applicable only to SQL Remote.
<b>Permissions</b>	Must have DBA authority.
<b>Side effects</b>	Automatic commit.
<b>See also</b>	CREATE SUBSCRIPTION statement SYNCHRONIZE SUBSCRIPTION statement "Synchronizing databases" in the chapter "SQL Remote Administration",
<b>Description</b>	<p>A SQL Remote subscription is said to be <b>started</b> when publication updates are being sent from the consolidated database to the remote database.</p> <p>The START SUBSCRIPTION statement is one of a set of statements that manage subscriptions. The CREATE SUBSCRIPTION statement defines the data that the subscriber is to receive. The SYNCHRONIZE SUBSCRIPTION statement ensures that the consolidated and remote databases are consistent with each other. The START SUBSCRIPTION statement is required to start messages being sent to the subscriber.</p> <p>Data at each end of the subscription must be consistent before a subscription is started. It is recommended that you use the database extraction utility to manage the creation, synchronization, and starting of subscriptions. If you use the database extraction utility, you do not need to execute an explicit START SUBSCRIPTION statement. Also, the Message Agent starts subscriptions once they are synchronized.</p>
<b>Example</b>	<p>The following statement starts the subscription of user <b>SamS</b> to the <b>pub_contact</b> publication.</p> <pre>START SUBSCRIPTION TO pub_contact FOR SamS</pre>

---

# STOP DATABASE statement

**Function** To stop a database on the specified database engine

**Syntax** **STOP DATABASE** *database-name*  
... [ **ON** *engine-name* ]  
... [ **UNCONDITIONALLY** ]

**Usage** ISQL

**Permissions** None

**Side effects** None

**See also** START DATABASE statement  
DISCONNECT statement

---

**Description** The STOP DATABASE statement stops a specified database on a specified database engine. If *engine-name* is not specified, all running engines will be searched for a database of the specified name.

If the UNCONDITIONALLY keyword is supplied, the database will be stopped even if there are connections to the database. If UNCONDITIONALLY is not specified, the database will not be stopped if there are connections to it.

**Examples** Stop the database named sample on the default engine.

```
STOP DATABASE sample ;
```

## **STOP ENGINE statement**

<b>Function</b>	To stop a database engine
<b>Syntax</b>	<b>STOP ENGINE</b> <i>engine-name</i> [ <b>UNCONDITIONALLY</b> ]
<b>Usage</b>	ISQL
<b>Permissions</b>	None
<b>Side effects</b>	None
<b>See also</b>	START ENGINE statement
<b>Description</b>	The STOP ENGINE statement stops the specified database engine. If the UNCONDITIONALLY keyword is supplied, the database engine will be stopped even if there are connections to the engine. If UNCONDITIONALLY is not specified, the database engine will not be stopped if there are connections to it.
<b>Example</b>	Stop the database engine named sample. <pre>STOP ENGINE sample</pre>

# STOP SUBSCRIPTION statement

<b>Function</b>	To stop a subscription for a user to a publication.
<b>Syntax</b>	<b>STOP SUBSCRIPTION TO</b> <i>publication-name</i> [ ( <i>constant</i> ) ] ... <b>FOR</b> <i>userid</i> ,...
<b>Usage</b>	Anywhere. This statement is applicable only to SQL Remote.
<b>Permissions</b>	Must have DBA authority.
<b>Side effects</b>	Automatic commit.
<b>See also</b>	CREATE SUBSCRIPTION statement SYNCHRONIZE SUBSCRIPTION statement "Synchronizing databases" in the chapter "SQL Remote Administration",
<b>Description</b>	<p>A SQL Remote subscription is said to be <b>started</b> when publication updates are being sent from the consolidated database to the remote database.</p> <p>The STOP SUBSCRIPTION statement prevents any further messages being sent to the subscriber. The START SUBSCRIPTION statement is required to restart messages being sent to the subscriber. However, you should ensure that the subscription is properly synchronized before restarting: that no messages have been missed.</p>
<b>Example</b>	<p>The following statement starts the subscription of user <b>SamS</b> to the <b>pub_contact</b> publication.</p> <pre>STOP SUBSCRIPTION TO pub_contact FOR SamS</pre>

## **SYNCHRONIZE SUBSCRIPTION statement**

<b>Function</b>	To synchronize a subscription for a user to a publication.
<b>Syntax</b>	<b>SYNCHRONIZE SUBSCRIPTION TO</b> <i>publication-name</i> [ ( <i>string</i> ) ] ... <b>FOR</b> <i>userid</i> ,...
<b>Usage</b>	Anywhere. This statement is applicable only to SQL Remote.
<b>Permissions</b>	Must have DBA authority.
<b>Side effects</b>	Automatic commit.
<b>See also</b>	CREATE SUBSCRIPTION statement START SUBSCRIPTION statement "Synchronizing databases" in the chapter "SQL Remote Administration",
<b>Description</b>	<p>A SQL Remote subscription is said to be <b>synchronized</b> when the data in the remote database is consistent with that in the consolidated database, so that publication updates sent from the consolidated database to the remote database will not result in conflicts and errors.</p> <p>To synchronize a subscription, a copy of the data in the publication at the consolidated database is sent to the remote database. The <b>SYNCHRONIZE SUBSCRIPTION</b> statement does this through the message system. It is recommended that where possible you use the database extraction utility instead to synchronize subscriptions without using a message system.</p>
<b>Example</b>	<p>The following statement synchronizes the subscription of user <b>SamS</b> to the <b>pub_contact</b> publication.</p> <pre>SYNCHRONIZE SUBSCRIPTION TO pub_contact FOR SamS</pre>

## SYSTEM statement

<b>Function</b>	To execute an operating system command from within ISQL.
<b>Syntax</b>	<b>SYSTEM</b> [ <i>operating-system-command</i> ]
<b>Usage</b>	ISQL (DOS and QNX only).
<b>Permissions</b>	None.
<b>Side effects</b>	None.
<b>See also</b>	COMMIT statement CONNECT statement
<b>Restrictions</b>	<ul style="list-style-type: none"><li>◆ The SYSTEM statement must be entirely contained on one line.</li><li>◆ Comments are not allowed at the end of a SYSTEM statement.</li></ul>
<b>Description</b>	Executes the specified operating system command. If no command is specified, the DOS command interpreter or QNX shell is started. You can return to ISQL by using the DOS <b>exit</b> command or by pressing CTRL+D in QNX.
<b>Example</b>	<pre>SYSTEM date</pre>

## **TRUNCATE TABLE statement**

<b>Function</b>	To delete all rows from a table, without deleting the table definition.
<b>Syntax</b>	<b>TRUNCATE TABLE</b> [ <i>owner.</i> ] <i>table-name</i>
<b>Usage</b>	Anywhere.
<b>Permissions</b>	Must be the table owner or have DBA authority.
<b>Side effects</b>	Delete triggers are not fired by the TRUNCATE TABLE statement.
<b>See also</b>	DELETE statement
<b>Description</b>	<p>The TRUNCATE TABLE statement deletes all rows from a table. It is equivalent to a DELETE statement without a WHERE clause, except that no triggers are fired as a result of the TRUNCATE TABLE statement and each individual row deletion is not entered into the transaction log.</p> <p>After a TRUNCATE TABLE statement, the table structure and all of the indexes continue to exist until you issue a DROP TABLE statement. The column definitions and constraints remain intact, and triggers and permissions remain in effect.</p> <p>The TRUNCATE TABLE statement is entered into the transaction log as a single statement, like data definition statements. Each deleted row is not entered into the transaction log.</p>
<b>Example</b>	<p>Delete all rows from the department table.</p> <pre>TRUNCATE TABLE department</pre>



# UNION operation

<b>Function</b>	To combine the results of two or more select statements.
<b>Syntax</b>	<i>select-without-order-by</i> ... <b>UNION</b> [ <b>ALL</b> ] <i>select-without-order-by</i> ... [ <b>UNION</b> [ <b>ALL</b> ] <i>select-without-order-by</i> ] ... ... [ <b>ORDER BY</b> <i>integer</i> [ <b>ASC</b>   <b>DESC</b> ], ... ]
<b>Usage</b>	Anywhere.
<b>Permissions</b>	Must have SELECT permission for each of the component SELECT statements.
<b>Side effects</b>	None.
<b>See also</b>	SELECT statement

**Description** The results of several SELECT statements can be combined into a larger result using UNION. The component SELECT statements must each have the same number of items in the select list, and cannot contain an ORDER BY clause.

The results of UNION ALL are the combined results of the component SELECT statements. The results of UNION are the same as UNION ALL except that duplicate rows are eliminated. Eliminating duplicates requires extra processing, so UNION ALL should be used instead of UNION where possible.

If corresponding items in two select lists have different data types, SQL Anywhere will choose a data type for the corresponding column in the result and automatically convert the columns in each component SELECT statement appropriately.

If ORDER BY is used, only integers are allowed in the order by list. These integers specify the position of the columns to be sorted.

The column names displayed are the same column names which would be displayed for the first SELECT statement.

**Examples** List all distinct surnames of employees and customers.

```
SELECT emp_lname
FROM Employee
UNION
SELECT lname
FROM Customer ;
```

## UNLOAD TABLE statement

**Function** To export data from a database table into an external ascii-format file.

**Syntax**

```
UNLOAD [ FROM ] TABLE [ owner ].table-name
... TO 'filename-string'
... [ FORMAT 'ascii' ] [ DELIMITED BY string ]
... [ QUOTES ON | OFF ] [ ESCAPES ON | OFF ]
... [ ORDER ON | OFF ]
... [ ESCAPE CHARACTER character ]
```

**Usage** Anywhere.

**Permissions** Must have SELECT permission on the table.

**Side effects** None.

**See also** LOAD TABLE statement  
OUTPUT statement

**Description** The UNLOAD TABLE statement allows efficient mass exporting from a database table into an ascii file. UNLOAD TABLE is more efficient than the ISQL statement OUTPUT and can be called from any client application.

UNLOAD TABLE places an exclusive lock on the whole table.

For descriptions of the FORMAT, DELIMITED BY, and ESCAPE CHARACTER options, see "LOAD TABLE statement" on page 1105. The other options are as follows:

**QUOTES option** With QUOTES turned on (the default), single quotes are placed around all exported strings.

**ESCAPES option** With ESCAPES on (the default), backslash-character combinations are used to identify special characters where necessary on export.

**ORDER option** With ORDER on (the default) the data is exported ordered by primary key values. With ORDER off, the data is exported in the same order you see when selecting from the table without an ORDER BY clause.

Exporting is slower with ORDER on. However, reloading using the LOAD TABLE statement is quicker because of the simplicity of the indexing step.

# UPDATE statement

<b>Function</b>	To modify data in the database.
<b>Syntax</b>	<p>Syntax 1.</p> <pre><b>UPDATE</b> <i>table-list</i> ... <b>SET</b> <i>column-name</i> = <i>expression</i>, ... ... [ <b>FROM</b> <i>table-list</i> ] ... [ <b>WHERE</b> <i>search-condition</i> ] ... [ <b>ORDER BY</b> <i>expression</i> [ <b>ASC</b>   <b>DESC</b> ] ,... ]</pre> <p>Syntax 2.</p> <pre><b>UPDATE</b> <i>table-list</i> ... <b>SET</b> <i>column-name</i> = <i>expression</i>, ... ... [ <b>VERIFY</b> ( <i>column-name</i>, ... ) <b>VALUES</b> ( <i>expression</i>, ... ) ] ... [ <b>WHERE</b> <i>search-condition</i> ] ... [ <b>ORDER BY</b> <i>expression</i> [ <b>ASC</b>   <b>DESC</b> ] ,... ]</pre> <p>Syntax 3:</p> <pre><b>UPDATE</b> <i>table</i> ...<b>PUBLICATION</b> <i>publication</i> ...{ <b>SUBSCRIBE BY</b> <i>expression</i> / <b>OLD SUBSCRIBE BY</b> <i>expression</i> <b>NEW SUBSCRIBE BY</b> <i>expression</i> } ...<b>WHERE</b> <i>search-condition</i></pre>
<b>Usage</b>	<p>Syntax 1 can be used Anywhere.</p> <p>.Syntax 2 and Syntax 3 are applicable only to SQL Remote. Syntax 3 is for use inside a BEFORE trigger.</p>
<b>Permissions</b>	Must have UPDATE permission for the columns being modified.
<b>Side effects</b>	None.
<b>See also</b>	DELETE statement INSERT statement FROM clause
<b>Description</b>	The UPDATE statement is used to modify rows of one or more tables. Each named column is set to the value of the expression on the right hand side of the equal sign. There are no restrictions on the <i>expression</i> . Even <i>column-name</i> can be used in the expression—the old value will be used. For a description of the table list and how to do joins, see "FROM clause" on page 1074

If no WHERE clause is specified, every row will be updated. If a WHERE clause is specified, then only those rows which satisfy the search condition will be updated.

Normally, the order that rows are updated doesn't matter. However, in conjunction with the NUMBER(\*) function, an ordering can be useful to get increasing numbers added to the rows in some specified order. Also, if you wish to do something like add 1 to the primary key values of a table, it is necessary to do this in descending order by primary key, so that you do not get duplicate primary keys during the operation.

Views can be updated provided the SELECT statement defining the view does not contain a GROUP BY clause, an aggregate function, or involve a UNION operation.

Character strings inserted into tables are always stored in the case they are entered, regardless of whether the database is case sensitive or not. Thus a character data type column updated with a string **Value** is always held in the database with an upper-case V and the remainder of the letters lower case. SELECT statements return the string as **Value**. If the database is not case-sensitive, however, all comparisons make **Value** the same as **value**, **VALUE**, and so on. Further, if a single-column primary key already contains an entry **Value**, an INSERT of **value** is rejected, as it would make the primary key not unique.

### Updates based on joins

The optional FROM clause allows tables to be updated based on joins. If the FROM clause is present, the WHERE clause qualifies the rows of the FROM clause. Data is updated only in the table list immediately following the UPDATE keyword.

If a FROM clause is used, it is important to qualify the table name that is being updated the same way in both parts of the statement. If a correlation name is used in one place, the same correlation name must be used in the other. Otherwise, an error is generated.

☞ For a full description of the FROM clause and joins, see "FROM clause" on page 1074.

### SQL Remote updates

Syntax 2 is intended for use with SQL Remote only, in single-row updates executed by the Message Agent. The VERIFY clause contains a set of values that are expected to be present in the row being updated. If the values do not match, any RESOLVE UPDATE triggers are fired before the UPDATE proceeds. The UPDATE does not fail if the VERIFY clause fails to match.

Syntax 3 is intended for use with SQL Remote only, inside a BEFORE trigger. The purpose is to provide a full list of subscribe by values any time the list changes. It is placed in SQL Remote triggers so that the database engine can compute the current list of subscribe by values. Both lists are placed in the transaction log.

The Message Agent uses the two lists to make sure that the row moves to any remote database that did not have the row and now needs it. The Message Agent also removes the row from any remote database that has the row and no longer needs it. A remote database that has the row and still needs it is not affected by the UPDATE statement.

Syntax 3 of the UPDATE statement allows the old subscribe by list and the new subscribe by list to be explicitly specified, which can make SQL Remote triggers more efficient. In the absence of these lists, the database engine computes the old subscribe by list from the publication definition. Since the new subscribe by list is commonly only slightly different from the old subscribe by list, the work to compute the old list may be done twice. By specifying both the old and new lists, this extra work can be avoided.

The SUBSCRIBE BY expression is either a value or a subquery.

### Using UPDATE to maintain subscriptions

Syntax 3 of the UPDATE statement is used to implement a specific SQL Remote feature, and is to be used inside a BEFORE trigger.

For publications created using a subquery in a SUBSCRIBE BY clause, you must write a trigger containing syntax 3 of the UPDATE statement in order to ensure that the rows are kept in their proper subscriptions.

☞ For a full description of this feature, see "Using subqueries in publications" in the chapter "SQL Remote Administration".

Syntax 3 of the UPDATE statement makes an entry in the transaction log, but does not change the database table.

### Examples

Transfer employee Philip Chin (employee 129) from the sales department to the marketing department.

```
UPDATE employee
SET dept_id = 400
WHERE emp_id = 129 ;
```

Sales orders currently start at ID 2001. Renumber all existing sales orders by subtracting 2000 from the id.

```
UPDATE sales_order_items AS items ,
sales_order AS orders
SET items.id = items.id - 2000,
orders.id = orders.id - 2000 ;
```

## UPDATE (positioned) statement

<b>Function</b>	To modify the data at the current location of a cursor.
<b>Syntax</b>	Syntax 1: <b>UPDATE WHERE CURRENT OF</b> <i>cursor-name</i> ... { <b>USING DESCRIPTOR</b> <i>sqlda-name</i>   <b>FROM</b> <i>host-variable-list</i> }  Syntax 2: <b>UPDATE</b> <i>table-list</i> ... <b>SET</b> <i>column-name</i> = <i>expression</i> , ... ... <b>WHERE CURRENT OF</b> <i>cursor-name</i>
<b>Parameters</b>	<i>host-variable-list</i> : indicator variables allowed <i>sqlda-name</i> : identifier
<b>Usage</b>	Embedded SQL, procedures, triggers, and batches..  The USING DESCRIPTOR, FROM <i>host-variable-list</i> , and <i>host-variable</i> formats are for Embedded SQL only.
<b>Permissions</b>	Must have UPDATE permission on the columns being modified.
<b>Side effects</b>	None.
<b>See also</b>	DELETE statement UPDATE statement
<b>Description</b>	<p>This form of the UPDATE statement updates the current row of the specified cursor. The current row is defined to be the last row FETCHed from the cursor and the last operation on the cursor must not have been a DELETE (positioned).</p> <p>For format 1, columns from the SQLDA or values from the host variable list correspond one-to-one with the select list items of the specified cursor. If the <b>sqldata</b> pointer in the SQLDA is the null pointer, then the corresponding select list item is not updated.</p> <p>In format 2, the requested columns are set to the specified values for the row at the current row of the specified query. The columns do not need to be in the select list of the specified open cursor. This format can be PREPARED.</p>
<b>Updating views</b>	Updates are not allowed on cursors on views that have more than one table in the FROM clause.
<b>Example</b>	<pre>UPDATE Employee SET emp_lname = 'Jones' WHERE CURRENT OF emp_cursor ;</pre>

## VALIDATE TABLE statement

---

<b>Function</b>	To validate a table in the database.
<b>Syntax</b>	<b>VALIDATE TABLE</b> [ <i>owner.</i> ] <i>table-name</i>
<b>Usage</b>	Anywhere.
<b>Permissions</b>	Must be the owner of the table or have DBA authority or have REMOTE DBA authority (SQL Remote).
<b>Side effects</b>	None.
<b>See also</b>	"The Validation utility" in the chapter "SQL Anywhere Components"
<b>Description</b>	<p>The VALIDATE TABLE statement will scan every row of a table and look each row up in each index on the table. If the database file is corrupt, an error will be reported. This should not happen. However, because DOS and Windows are unprotected operating environments, other software can corrupt memory used by the database engine. This problem may be detected through software errors or crashes, or the corrupt memory could get written to the database creating a corrupt database file. Also, in any operating system, hardware problems with the disk could cause the database file to get corrupted.</p> <p>If you do have errors reported, you can drop all of the indexes and keys on a table and recreate them. Any foreign keys to the table will also need to be recreated. Another solution to errors reported by VALIDATE TABLE is to unload and reload your entire database. You should use the -u option of DBUNLOAD so that it will not try to use a possibly corrupt index to order the data.</p>

---

## WHENEVER statement

**Function** To specify error handling in an Embedded SQL program.

**Syntax** **WHENEVER**  
**SQLERROR**  
**| SQLWARNING**  
**| NOTFOUND**

...  
**GOTO** *label*  
**| STOP**  
**| CONTINUE**  
**| { C code; }**

**Parameters** *label: identifier*

**Usage** Embedded SQL.

**Permissions** None.

**Side effects** None.

**Description** The WHENEVER statement is used to trap errors, warnings and exceptional conditions encountered by the database when processing SQL statements. The statement can be put anywhere in an Embedded SQL C program and does not generate any code. The preprocessor will generate code following each successive SQL statement. The error action remains in effect for all Embedded SQL statements from the source line of the WHENEVER statement until the next WHENEVER statement with the same error condition, or the end of the source file.

### Errors based on source position

The error conditions are in effect based on positioning in the C language source file and not on when the statements are executed.

The default action is CONTINUE.

Note that this statement is provided for convenience in simple programs. Most of the time, checking the sqlcode field of the SQLCA (SQLCODE) directly is the easiest way to check error conditions. In this case, the WHENEVER statement would not be used. In fact, all the WHENEVER statement does is cause the preprocessor to generate an *if ( SQLCODE )* test after each statement.

**Examples** The following are examples of the WHENEVER statement:

```
EXEC SQL WHENEVER NOTFOUND GOTO done;
```



```
EXEC SQL WHENEVER SQLERROR  
{ PrintError( &sqlca ); return( FALSE ); };
```

---



## CHAPTER 42

# SQL Anywhere Database Error Messages

### About this chapter

This chapter lists all database error messages reported by SQL Anywhere. Many of the errors contain the characters %1, %2 and so on. These are replaced by the parameters to the error message.

Each error has a numeric error code, called the SQLCODE. Negative codes are considered errors; positive codes are warnings. The SQLCODE 0 indicates successful completion.

The full listing of the error messages is ordered by SQLCODE value. To find an error message description if you do not have the SQLCODE value, look up the SQLCODE in the alphabetic or SQLSTATE listing.

### Contents

<b>Topic</b>	<b>Page</b>
Error message index by SQLCODE	1193
Error messages index by SQLSTATE	1201
Alphabetic list of error messages	1210
Internal errors (assertion failed)	1296

## Error message index by SQLCODE

SQLCODE	Message	Page
-642	Invalid SQL descriptor name	1249
-641	Error in assignment	1234
-640	Invalid descriptor index	1243
-639	Parameter name missing in call to procedure '%1'	1260
-638	Right truncation of string data	1266
-637	Duplicate insert column	1233
-636	Duplicate referencing column	1233
-635	GRANT of column permission on view not allowed	1237
-634	Unterminated C string	1286
-633	Update operation attempted on a read-only cursor	1286
-632	WITH CHECK OPTION violated for view '%1'	1293
-631	RAISERROR executed: %1	1264
-630	Invalid escape sequence '%1'	1243
-629	Invalid escape character '%1'	1243
-628	Division by zero	1232
-627	Disallowed language extension detected in syntax near '%1'	1231
-626	A thread used internally could not be started	1210
-625	Too many parameters to this external procedure call	1281
-624	A parameter to an external function is an unsupported datatype	1210
-623	Data definition statements not allowed in procedures or triggers	1227
-622	Could not allocate resources to call external function	1224
-621	Could not find the named function in the dynamic library	1225
-620	Could not load the dynamic library	1225
-619	Need a dynamic library name	1254
-618	Mismatch between external function platform	1253

SQLCODE	Message	Page
	specifier and current operating system	
-617	Calling functions outside the database engine is not supported	1213
-616	Too many columns in table	1280
-615	Parameter '%1' not found in procedure '%2'	1260
-614	Cannot drop a user that owns messages or datatypes	1215
-613	User-defined type %1 not found	1290
-612	User message %1 not found	1290
-611	Transact-SQL feature not supported	1281
-610	User message %1 already exists	1289
-609	Invalid datatype for column in WRITETEXT or READTEXT	1242
-608	Invalid TEXTPTR value used with WRITETEXT or READTEXT	1249
-407	An argument passed to a SQL Anywhere HLI function was invalid	1212
-406	SQL Anywhere HLI internal error	1270
-405	Invalid SQL Anywhere HLI callback function	1247
-404	Invalid SQL Anywhere HLI host variable value	1248
-403	Invalid SQL Anywhere HLI host variable name	1248
-402	Invalid SQL Anywhere HLI statement name	1248
-401	Invalid SQL Anywhere HLI cursor name	1247
-400	Invalid SQL Anywhere HLI command syntax	1247
-312	User '%1' already has membership in group '%2'	1287
-308	Connection was terminated	1223
-307	All threads are blocked	1211
-306	Deadlock detected	1231
-305	I/O error %1 -- transaction rolled back	1238
-304	Disk full '%1' -- transaction rolled back	1232
-302	Terminated by user -- transaction rolled back	1277
-301	Internal database error %1 -- transaction rolled back	1241

SQLCODE	Message	Page
-300	Run time SQL error -- %1	1268
-299	Statement interrupted by user	1272
-298	Attempted two active database requests	1212
-297	User-defined exception signalled	1290
-296	Error number %1 for RAISERROR is less than 17000	1234
-295	Cannot uniquely identify rows in cursor	1218
-294	Format string argument number %1 is invalid	1236
-288	Remote statement failed	1265
-287	Passthrough statement inconsistent with current passthrough	1261
-286	Remote message type '%1' not found	1264
-285	User '%1' is not a remote user for this database	1288
-284	User '%1' is already the publisher for this database	1288
-283	Subscription to '%1' for '%2' not found	1273
-282	Subscription to '%1' for '%2' already exists	1273
-281	Table '%1' has publications	1275
-280	Publication '%1' not found	1264
-275	Triggers and procedures not supported in runtime engine	1283
-274	Procedure or trigger calls have nested too deeply	1263
-273	COMMIT/ROLLBACK not allowed within trigger actions	1222
-272	Invalid REFERENCES clause in trigger definition	1246
-271	Trigger definition conflicts with existing triggers	1283
-270	Cannot drop a user that owns procedures in runtime engine	1216
-269	Cannot delete a column referenced in a trigger definition	1215
-268	Trigger '%1' not found	1282
-267	COMMIT/ROLLBACK not allowed within atomic operation	1222
-266	Database was initialized with an older version of	1229

SQLCODE	Message	Page
	the software	
-265	Procedure '%1' not found	1262
-264	Wrong number of variables in FETCH	1294
-263	Invalid absolute or relative offset in FETCH	1241
-262	Label '%1' not found	1252
-261	There is already a variable named '%1'	1279
-260	Variable '%1' not found	1293
-251	Foreign key '%1' for table '%2' duplicates an existing foreign key	1236
-250	Identifier '%1' too long	1238
-245	Integrated logon failed	1240
-244	Transaction log was truncated	1282
-243	Unable to delete database file	1283
-242	Incomplete transactions prevent transaction log renaming	1238
-241	Database backup not started	1227
-240	Unknown backup operation	1285
-232	Server/database engine version mismatch	1269
-231	Dbllib/database engine version mismatch	1230
-230	Sqlpp/dbllib version mismatch	1271
-222	Result set not allowed from within an atomic compound statement	1266
-221	ROLLBACK TO SAVEPOINT not allowed	1266
-220	Savepoint '%1' not found	1268
-215	Procedure in use	1263
-214	Table in use	1277
-213	Savepoints require a rollback log	1268
-212	CHECKPOINT command requires a rollback log	1219
-211	Not allowed while %1 is using the database	1256
-210	User '%1' has the row in '%2' locked	1288
-209	Invalid value for column '%1' in table '%2'	1251

SQLCODE	Message	Page
-208	Row has changed since last read -- operation cancelled	1267
-207	Wrong number of values for INSERT	1294
-206	Standard logons are not permitted	1271
-205	Integrated logons are not permitted	1240
-204	Only the DBA can set the option %1	1259
-203	Cannot set a temporary option for user '%1'	1218
-202	Only PUBLIC settings are allowed for option '%1'	1259
-201	Invalid setting for option '%1'	1246
-200	Invalid option '%1' -- no PUBLIC setting exists	1245
-199	INSERT/DELETE on cursor can modify only one table	1240
-198	Primary key for row in table '%1' is referenced in another table	1261
-197	No current row of cursor	1255
-196	Index '%1' for table '%2' would not be unique	1239
-195	Column '%1' in table '%2' cannot be NULL	1221
-194	No primary key value for foreign key '%1' in table '%2'	1256
-193	Primary key for table '%1' is not unique	1262
-192	Update operation attempted on non-updatable query	1286
-191	Cannot modify column '%1' in table '%2'	1217
-190	Cannot update an expression	1219
-189	Unable to find in index '%1' for table '%2'	1284
-188	Not enough values for host variables	1257
-187	Invalid operation for this cursor	1245
-186	Subquery cannot return more than one result	1273
-185	SELECT returns more than one row	1269
-184	Error inserting into cursor	1234
-183	Cannot find index named '%1'	1216
-182	Not enough fields allocated in SQLDA	1257



SQLCODE	Message	Page
-181	No indicator variable provided for NULL result	1255
-180	Cursor not open	1226
-172	Cursor already open	1225
-171	Error opening cursor	1235
-170	Cursor has not been declared	1226
-162	Cannot outer join a view with a UNION or GROUP BY	1218
-161	Invalid type on DESCRIBE statement	1250
-160	Can only describe a SELECT statement	1213
-159	Invalid column number	1241
-158	Value %1 out of range for destination	1291
-157	Cannot convert %1 to a %2	1215
-156	Invalid expression near '%1'	1244
-155	Invalid host variable	1244
-154	Wrong number of parameters to function '%1'	1294
-153	SELECT lists in UNION do not match in length	1269
-152	Number in ORDER BY is too large	1258
-151	Subquery allowed only one select list item	1272
-150	Aggregate functions not allowed on this statement	1211
-149	Function or column reference to '%1' in the select list must also appear in a GROUP BY	1237
-148	Unknown function '%1'	1285
-147	There is more than one way to join '%1' to '%2'	1279
-146	There is no way to join '%1' to '%2'	1280
-145	Foreign key name '%1' not found	1236
-144	Column '%1' found in more than one table -- need a correlation name	1220
-143	Column '%1' not found	1221
-142	Correlation name '%1' not found	1224
-141	Table '%1' not found	1275
-140	Userid '%1' does not exist	1291

SQLCODE	Message	Page
-139	More than one table is identified as '%1'	1254
-138	DbSPACE '%1' not found	1230
-137	Table '%1' requires a unique correlation name	1276
-136	Table '%1' is in an outer join cycle	1275
-135	Language extension	1252
-134	Feature '%1' not implemented	1235
-133	Invalid prepared statement type	1246
-132	SQL statement error	1271
-131	Syntax error near '%1'	1274
-130	Invalid statement	1249
-128	Cannot drop a user that owns tables in runtime engine	1216
-127	Cannot alter a column in an index	1214
-126	Table cannot have two primary keys	1276
-125	ALTER clause conflict	1212
-124	More columns are being dropped from table %1 than are defined	1253
-123	User '%1' is not a user group	1289
-122	Operation would cause a group cycle	1260
-121	Do not have permission to %1	1232
-120	User '%1' already has grant permission	1287
-119	Primary key column '%1' already defined	1261
-118	Table '%1' has no primary key	1274
-116	Table must be empty	1277
-114	Number of columns does not match SELECT	1259
-113	Column %1 in foreign key has a different definition than primary key	1220
-112	Table already has a primary key	1276
-111	Index name '%1' not unique	1239
-110	Item '%1' already exists	1251
-109	There are still active database connections	1279

SQLCODE	Message	Page
-108	Connection not found	1223
-107	Error writing to log file	1235
-106	Cannot open log file %1	1217
-105	Cannot be started -- %1	1214
-104	Invalid userid and password on preprocessed module	1250
-103	Invalid userid or password	1251
-102	Too many connections to database	1281
-101	Not connected to SQL database	1256
-100	Database engine not running	1228
-99	Connections to database have been disabled	1224
-98	Authentication violation	1213
-97	Database's page size too big	1230
-96	Database engine already running	1227
-95	Invalid parameter	1245
-89	Database engine not running in multi-user mode	1228
-88	Client/server communications protocol mismatch	1220
-87	Database name required to start engine	1229
-86	Not enough memory to start	1257
-85	Communication error	1223
-84	Specified database is invalid	1270
-83	Specified database not found	1270
-82	Unable to start specified database	1285
-81	Invalid database engine command line	1242
-80	Unable to start database engine	1284
-79	Invalid local database switch	1244
-78	Dynamic memory exhausted!	1233
-77	Database name not unique	1228
-76	Request denied -- no active databases	1265
-75	Request to start/stop database denied	1265

<b>SQLCODE</b>	<b>Message</b>	<b>Page</b>
-74	The selected database is currently inactive	1278
-73	Communication buffer underflow	1222
0	(no message)	1210
100	No data	1255
101	Value truncated	1292
102	Using temporary table	1291
103	Invalid data conversion	1242
104	Row has been updated since last time read	1267
105	Procedure has completed	1263
106	Value for column '%1' in table '%2' has changed	1292
107	Language extension detected in syntax	1252
108	Cursor operation conflict	1226
109	Null value eliminated in aggregate function	1258
110	Transaction log backup page only partially full	1282
111	Statement cannot be executed	1272
112	More info required	1254
113	Database option '%1' for user '%2' has an invalid setting	1229
200	Warning	1293
400	The supplied buffer was too small to hold all requested query results	1278

## Error messages index by SQLSTATE

SQL Anywhere supports the SQLSTATE error code defined by SQL/92. Each SQLSTATE value is a 5-character string containing a 2-character class followed by a 3-character subclass. Each character can be one of the uppercase letters A through Z or the digits 0 through 9. A class that begins with A through H or 0 through 4 has been defined by the ANSI standard; other classes are implementation defined. Similarly, subclasses of standard classes that start with the same characters (A-H, 0-4) are standard. The subclass 000 always means that no subclass code is defined. The most common SQLSTATE value is 00000, which indicates successful completion.

SQLSTATE	Message	Page
00000	(no message)	1210
01000	Warning	1293
01001	Cursor operation conflict	1226
01003	Null value eliminated in aggregate function	1258
01004	Value truncated	1292
01W02	Using temporary table	1291
01W03	Invalid data conversion	1242
01W04	Row has been updated since last time read	1267
01W05	Procedure has completed	1263
01W06	Value for column '%1' in table '%2' has changed	1292
01W07	Language extension detected in syntax	1252
01W08	Statement cannot be executed	1272
01W09	More info required	1254
01W10	Transaction log backup page only partially full	1282
01W11	Database option '%1' for user '%2' has an invalid setting	1229
01WH1	The supplied buffer was too small to hold all requested query results	1278
02W01	No data	1255
07001	Not enough values for host variables	1257
07002	Not enough fields allocated in SQLDA	1257

<b>SQLSTATE</b>	<b>Message</b>	<b>Page</b>
07003	Error opening cursor	1235
07005	Can only describe a SELECT statement	1213
07009	Invalid descriptor index	1243
07W01	Invalid type on DESCRIBE statement	1250
07W02	Invalid statement	1249
07W03	Invalid prepared statement type	1246
08001	Cannot be started -- %1	1214
08003	Not connected to SQL database	1256
08004	Userid '%1' does not exist	1291
08W01	Database engine not running	1228
08W02	Connection not found	1223
08W03	Too many connections to database	1281
08W04	Connections to database have been disabled	1224
08W05	Cannot open log file %1	1217
08W06	There are still active database connections	1279
08W07	Unable to start database engine	1284
08W08	Invalid database engine command line	1242
08W09	Unable to start specified database	1285
08W10	Specified database not found	1270
08W11	Specified database is invalid	1270
08W12	Communication error	1223
08W13	Not enough memory to start	1257
08W14	Database name required to start engine	1229
08W15	Client/server communications protocol mismatch	1220
08W16	Database engine not running in multi-user mode	1228
08W17	Error writing to log file	1235
08W18	Sqlpp/dblib version mismatch	1271
08W19	Dblib/database engine version mismatch	1230
08W20	Server/database engine version mismatch	1269
08W21	Authentication violation	1213

SQLSTATE	Message	Page
08W22	Database's page size too big	1230
08W23	Database engine already running	1227
08W24	Invalid parameter	1245
08W25	Invalid local database switch	1244
08W26	Dynamic memory exhausted!	1233
08W27	Database name not unique	1228
08W28	Request denied -- no active databases	1265
08W29	Request to start/stop database denied	1265
08W30	The selected database is currently inactive	1278
08W31	Communication buffer underflow	1222
09W01	Error inserting into cursor	1234
09W02	Invalid operation for this cursor	1245
09W04	INSERT/DELETE on cursor can modify only one table	1240
09W05	Cannot uniquely identify rows in cursor	1218
0A000	Feature '%1' not implemented	1235
0AW01	Language extension	1252
0AW02	Transact-SQL feature not supported	1281
0AW03	Disallowed language extension detected in syntax near '%1'	1231
0AW04	Triggers and procedures not supported in runtime engine	1283
21000	SELECT returns more than one row	1269
21W01	Subquery cannot return more than one result	1273
22001	Right truncation of string data	1266
22002	No indicator variable provided for NULL result	1255
22003	Value %1 out of range for destination	1291
22005	Error in assignment	1234
22012	Division by zero	1232
22019	Invalid escape character '%1'	1243
22024	Unterminated C string	1286

SQLSTATE	Message	Page
22025	Invalid escape sequence '%1'	1243
22W01	An argument passed to a SQL Anywhere HLI function was invalid	1212
22W02	Row has changed since last read -- operation cancelled	1267
22W03	Invalid TEXTPTR value used with WRITETEXT or READTEXT	1249
23502	Column '%1' in table '%2' cannot be NULL	1221
23503	No primary key value for foreign key '%1' in table '%2'	1256
23505	Index '%1' for table '%2' would not be unique	1239
23506	Invalid value for column '%1' in table '%2'	1251
23W01	Primary key for table '%1' is not unique	1262
23W05	Primary key for row in table '%1' is referenced in another table	1261
24501	Cursor not open	1226
24502	Cursor already open	1225
24503	No current row of cursor	1255
24W01	Cursor has not been declared	1226
26501	SQL statement error	1271
26W01	Invalid SQL Anywhere HLI statement name	1248
28000	Invalid userid or password	1251
28W01	Invalid userid and password on preprocessed module	1250
28W02	Integrated logons are not permitted	1240
28W03	Standard logons are not permitted	1271
28W04	Integrated logon failed	1240
2D501	COMMIT/ROLLBACK not allowed within trigger actions	1222
33000	Invalid SQL descriptor name	1249
34W01	Invalid SQL Anywhere HLI cursor name	1247
37505	Wrong number of parameters to function '%1'	1294
3B001	Savepoint '%1' not found	1268



SQLSTATE	Message	Page
3B002	ROLLBACK TO SAVEPOINT not allowed	1266
3BW01	Savepoints require a rollback log	1268
3BW02	Result set not allowed from within an atomic compound statement	1266
40000	Run time SQL error -- %1	1268
40001	Deadlock detected	1231
40W01	Internal database error %1 -- transaction rolled back	1241
40W02	Terminated by user -- transaction rolled back	1277
40W03	Disk full '%1' -- transaction rolled back	1232
40W04	I/O error %1 -- transaction rolled back	1238
40W06	All threads are blocked	1211
40W07	Connection was terminated	1223
42501	Do not have permission to %1	1232
42W01	User '%1' already has grant permission	1287
42W02	Operation would cause a group cycle	1260
42W03	User '%1' is not a user group	1289
42W04	Syntax error near '%1'	1274
42W05	Unknown function '%1'	1285
42W06	Aggregate functions not allowed on this statement	1211
42W07	Invalid host variable	1244
42W08	Invalid expression near '%1'	1244
42W09	Invalid SQL Anywhere HLI host variable name	1248
42W10	Invalid SQL Anywhere HLI host variable value	1248
42W11	Invalid SQL Anywhere HLI command syntax	1247
42W12	Invalid SQL Anywhere HLI callback function	1247
42W13	Invalid column number	1241
42W14	Variable '%1' not found	1293
42W15	There is already a variable named '%1'	1279
42W16	Invalid option '%1' -- no PUBLIC setting exists	1245
42W17	Invalid setting for option '%1'	1246

SQLSTATE	Message	Page
42W18	User '%1' has the row in '%2' locked	1288
42W19	Not allowed while %1 is using the database	1256
42W20	CHECKPOINT command requires a rollback log	1219
42W21	Table in use	1277
42W22	Attempted two active database requests	1212
42W23	Procedure in use	1263
42W24	Label '%1' not found	1252
42W25	Invalid absolute or relative offset in FETCH	1241
42W26	Wrong number of variables in FETCH	1294
42W27	Database was initialized with an older version of the software	1229
42W28	COMMIT/ROLLBACK not allowed within atomic operation	1222
42W29	Procedure or trigger calls have nested too deeply	1263
42W30	Update operation attempted on a read-only cursor	1286
42W31	Update operation attempted on non-updatable query	1286
42W32	Cannot modify column '%1' in table '%2'	1217
42W33	Table '%1' not found	1275
42W34	User '%1' already has membership in group '%2'	1287
42W40	Duplicate referencing column	1233
42W41	Duplicate insert column	1233
42W42	Parameter name missing in call to procedure '%1'	1260
42W43	Only PUBLIC settings are allowed for option '%1'	1259
42W44	More columns are being dropped from table %1 than are defined	1253
42W45	Cannot set a temporary option for user '%1'	1218
42W46	Only the DBA can set the option %1	1259
42W47	Parameter '%1' not found in procedure '%2'	1260
44000	WITH CHECK OPTION violated for view '%1'	1293
52002	Column '%1' found in more than one table -- need a correlation name	1220

SQLSTATE	Message	Page
52003	Column '%1' not found	1221
52009	Primary key column '%1' already defined	1261
52010	Item '%1' already exists	1251
52012	More than one table is identified as '%1'	1254
52W02	Correlation name '%1' not found	1224
52W03	Cannot find index named '%1'	1216
52W04	Index name '%1' not unique	1239
52W05	Table cannot have two primary keys	1276
52W06	Foreign key '%1' for table '%2' duplicates an existing foreign key	1236
52W07	Foreign key name '%1' not found	1236
52W08	There is more than one way to join '%1' to '%2'	1279
52W09	Procedure '%1' not found	1262
52W10	Trigger '%1' not found	1282
52W11	Trigger definition conflicts with existing triggers	1283
52W12	Invalid REFERENCES clause in trigger definition	1246
52W13	DbSPACE '%1' not found	1230
52W14	Table '%1' is in an outer join cycle	1275
52W15	Table '%1' requires a unique correlation name	1276
52W16	User message %1 already exists	1289
52W17	User message %1 not found	1290
52W18	User-defined type %1 not found	1290
52W19	Cannot outer join a view with a UNION or GROUP BY	1218
52W20	Too many columns in table	1280
52W21	Data definition statements not allowed in procedures or triggers	1227
52W22	GRANT of column permission on view not allowed	1237
53002	Wrong number of values for INSERT	1294
53003	Function or column reference to '%1' in the select list must also appear in a GROUP BY	1237

## Error messages index by SQLSTATE

---

SQLSTATE	Message	Page
53005	Number in ORDER BY is too large	1258
53011	Number of columns does not match SELECT	1259
53018	Cannot convert %1 to a %2	1215
53023	Subquery allowed only one select list item	1272
53026	SELECT lists in UNION do not match in length	1269
53030	Column %1 in foreign key has a different definition than primary key	1220
53W01	ALTER clause conflict	1212
53W02	Cannot update an expression	1219
53W04	There is no way to join '%1' to '%2'	1280
53W05	Cannot alter a column in an index	1214
53W06	Cannot delete a column referenced in a trigger definition	1215
53W07	Error number %1 for RAISERROR is less than 17000	1234
53W08	Format string argument number %1 is invalid	1236
53W09	Invalid datatype for column in WRITETEXT or READTEXT	1242
54003	Identifier '%1' too long	1238
55008	Table '%1' has no primary key	1274
55013	Table already has a primary key	1276
55W02	Table must be empty	1277
55W03	Cannot drop a user that owns tables in runtime engine	1216
55W04	Cannot drop a user that owns procedures in runtime engine	1216
55W05	Cannot drop a user that owns messages or datatypes	1215
57014	Statement interrupted by user	1272
5RW01	Publication '%1' not found	1264
5RW02	Table '%1' has publications	1275
5RW03	Subscription to '%1' for '%2' already exists	1273
5RW04	Subscription to '%1' for '%2' not found	1273

SQLSTATE	Message	Page
5RW05	User '%1' is already the publisher for this database	1288
5RW06	User '%1' is not a remote user for this database	1288
5RW07	Remote message type '%1' not found	1264
5RW08	Passthrough statement inconsistent with current passthrough	1261
5RW09	Remote statement failed	1265
99999	User-defined exception signalled	1290
WB001	Unknown backup operation	1285
WB002	Database backup not started	1227
WB003	Incomplete transactions prevent transaction log renaming	1238
WB004	Unable to delete database file	1283
WB005	Transaction log was truncated	1282
WI005	Unable to find in index '%1' for table '%2'	1284
WI007	SQL Anywhere HLI internal error	1270
WW003	Calling functions outside the database engine is not supported	1213
WW004	Mismatch between external function platform specifier and current operating system	1253
WW005	Need a dynamic library name	1254
WW006	Could not load the dynamic library	1225
WW007	Could not find the named function in the dynamic library	1225
WW008	Could not allocate resources to call external function	1224
WW009	A parameter to an external function is an unsupported datatype	1210
WW010	Too many parameters to this external procedure call	1281
WW011	A thread used internally could not be started	1210
WW012	RAISERROR executed: %1	1264

## Alphabetic list of error messages

This section provides a full listing of error messages and descriptions.

Errors with an ODBC state marked "handled by ODBC driver" are not returned to ODBC applications, as the ODBC driver carries out the required actions.

### (no message)

Item	Value
SQLCode	0
Constant	SQLE_NOERROR
SQLState	00000
ODBC State	00000

**Probable cause** This code indicates that there was no error or warning.

### A parameter to an external function is an unsupported datatype

Item	Value
SQLCode	-624
Constant	SQLE_DATATYPE_NOT_ALLOWED
SQLState	WW009
ODBC State	S1000

**Probable cause** A parameter to a call to an external function is declared to be a datatype that is not supported.

### A thread used internally could not be started

Item	Value
SQLCode	-626
Constant	SQLE_THREAD_START_FAILURE

Item	Value
SQLState	WW011
ODBC State	S1000

**Probable cause**

This is a Windows95 specific error. An operating system thread could not be started that is required to execute external functions.

**Aggregate functions not allowed on this statement**

Item	Value
SQLCode	-150
Constant	SQLE_AGGREGATES_NOT_ALLOWED
SQLState	42W06
ODBC State	37000

**Probable cause**

An UPDATE statement has used an aggregate function (MIN, MAX, SUM, AVG or COUNT).

**All threads are blocked**

Item	Value
SQLCode	-307
Constant	SQLE_THREAD_DEADLOCK
SQLState	40W06
ODBC State	40001

**Probable cause**

You have attempted to read or write a row and it is locked by another user. Also, all other threads (see database option `THREAD_COUNT`) are blocked waiting for a lock to be released. This is a deadlock situation and your transaction has been chosen as the one to rollback.

## ALTER clause conflict

Item	Value
SQLCode	-125
Constant	SQLE_ALTER_CLAUSE_CONFLICT
SQLState	53W01
ODBC State	S1000

**Probable cause** A primary key clause, foreign key clause, or a uniqueness clause must be the only clause of an ALTER TABLE command.

## An argument passed to a SQL Anywhere HLI function was invalid

Item	Value
SQLCode	-407
Constant	SQLE_HLI_BAD_ARGUMENT
SQLState	22W01
ODBC State	22W01

**Probable cause** One of the arguments passed to a WSQL HLI function was invalid. This may indicate that a pointer to a command string or result buffer is the null pointer.

## Attempted two active database requests

Item	Value
SQLCode	-298
Constant	SQLE_DOUBLE_REQUEST
SQLState	42W22
ODBC State	S1000



**Probable cause**

In Embedded SQL, you have attempted to submit a database request while you have another request in process. This often occurs in Windows when processing the WM\_PAINT message causes a database request, and you get a second WM\_PAINT before the database request has completed.

**Authentication violation**

Item	Value
SQLCode	-98
Constant	SQLE_AUTHENTICATION_VIOLATION
SQLState	08W21
ODBC State	08001

**Probable cause**

You have attempted to connect to an engine or server which has been authenticated for exclusive use with a specific application.

**Calling functions outside the database engine is not supported**

Item	Value
SQLCode	-617
Constant	SQLE_EXTERNAL_CALLS_NOT_SUPPORTED
SQLState	WW003
ODBC State	S1000

**Probable cause**

An attempt was made to call a stored procedure that, in turn, calls a function in a dynamically loaded module. The operating system on which this stored procedure was called does not support such an action.

**Can only describe a SELECT statement**

Item	Value
SQLCode	-160
Constant	SQLE_DESCRIBE_NONSELECT

Item	Value
SQLState	07005
ODBC State	(handled by ODBC driver)

**Probable cause** In the C language interface, you attempted to describe the select list of a statement other than a SELECT statement.

## Cannot alter a column in an index

Item	Value
SQLCode	-127
Constant	SQLE_COLUMN_IN_INDEX
SQLState	53W05
ODBC State	S1000

**Probable cause** This error is reported if you attempt to delete or modify the definition of a column that is part of a primary or foreign key. This error will also be reported if you attempt to delete a column that has an index on it. DROP the index or key, perform the ALTER command, and then add the index or key again.

## Cannot be started -- %1

Item	Value
SQLCode	-105
Constant	SQLE_UNABLE_TO_CONNECT
SQLState	08001
ODBC State	08001
Parameter 1	Name of database.

**Probable cause** The specified database environment cannot be found. If it is a database name, then it does not exist, it is not a database, it is corrupt, or it is an older format. If it is a server name, then the server cannot be found.

**Cannot convert %1 to a %2**

Item	Value
SQLCode	-157
Constant	SQLE_CONVERSION_ERROR
SQLState	53018
ODBC State	07006
Parameter 1	The value that could not be converted.
Parameter 2	The name of the type for the conversion.

**Probable cause**

An invalid value has been supplied to or fetched from the database. For example, the value 12X might have been supplied where a number was required.

**Cannot delete a column referenced in a trigger definition**

Item	Value
SQLCode	-269
Constant	SQLE_COLUMN_IN_TRIGGER
SQLState	53W06
ODBC State	S1000

**Probable cause**

This error is reported if you attempt to delete a column that is referenced in a trigger definition. DROP the trigger before performing the ALTER command.

**Cannot drop a user that owns messages or datatypes**

Item	Value
SQLCode	-614
Constant	SQLE_USER_OWNS_MESSAGES_OR_DATATYPES
SQLState	55W05
ODBC State	37000

**Probable cause** A user to be dropped is the creator of a message or user-defined datatype. The message or user-defined datatype must be dropped first.

### Cannot drop a user that owns procedures in runtime engine

Item	Value
SQLCode	-270
Constant	SQLE_USER_OWNS_PROCEDURES
SQLState	55W04
ODBC State	37000

**Probable cause** This error is reported by the runtime engine if you attempt to drop a user that owns procedures. Because this operation would result in dropping procedures, and the runtime engine cannot drop procedures, it is not allowed. Use the full engine.

### Cannot drop a user that owns tables in runtime engine

Item	Value
SQLCode	-128
Constant	SQLE_USER_OWNS_TABLES
SQLState	55W03
ODBC State	37000

**Probable cause** This error is reported by the runtime engine if you attempt to drop a user that owns tables. Because this operation would result in dropping tables, and the runtime engine cannot drop tables, it is not allowed. Use the full engine.

### Cannot find index named '%1'

Item	Value
SQLCode	-183
Constant	SQLE_INDEX_NOT_FOUND

Item	Value
SQLState	52W03
ODBC State	S0012
Parameter 1	Name of the index that cannot be found.

**Probable cause**

A DROP INDEX command has named an index that does not exist. Check for spelling errors or whether the index name must be qualified by a userid.

**Cannot modify column '%1' in table '%2'**

Item	Value
SQLCode	-191
Constant	SQLE_CANNOT_MODIFY
SQLState	42W32
ODBC State	42032
Parameter 1	Name of the column that cannot be changed.
Parameter 2	Name of the table containing the column.

**Probable cause**

You do not have permission to modify the column, or the table is actually a view and the column in the view is defined as an expression (such as column1+column2) that cannot be modified.

**Cannot open log file %1**

Item	Value
SQLCode	-106
Constant	SQLE_CANNOT_OPEN_LOG
SQLState	08W05
ODBC State	08003
Parameter 1	Name of log file.

**Probable cause** The database engine was unable to open the transaction log file. Perhaps the log file name specifies an invalid device or directory. If this is the case, you can use the DBLOG utility to find out where the transaction log should be and perhaps change it.

## Cannot outer join a view with a UNION or GROUP BY

Item	Value
SQLCode	-162
Constant	SQLE_CANNOT_OUTER_JOIN
SQLState	52W19
ODBC State	37000

**Probable cause** A view that contains a UNION or view cannot be used on the right of a LEFT OUTER JOIN or on the left of a RIGHT OUTER JOIN.

## Cannot set a temporary option for user '%1'

Item	Value
SQLCode	-203
Constant	SQLE_TEMPORARY_NOT_ALLOWED
SQLState	42W45
ODBC State	37000
Parameter 1	Userid whose option was to be changed.

**Probable cause** TEMPORARY options are set on a connection basis. To change an option for another user, do not specify TEMPORARY in the SET OPTION statement.

## Cannot uniquely identify rows in cursor

Item	Value
SQLCode	-295

Item	Value
Constant	SQLE_CANNOT_UNIQUELY_IDENTIFY_ROWS
SQLState	09W05
ODBC State	24000

**Probable cause**

A UNIQUE cursor has been opened on a SELECT statement for which a set of columns uniquely identifying each row cannot be generated. One of the tables may not be defined with a primary key or uniqueness constraint, or the SELECT statement may involve a UNION or GROUP BY.

**Cannot update an expression**

Item	Value
SQLCode	-190
Constant	SQLE_NON_UPDATEABLE_COLUMN
SQLState	53W02
ODBC State	37000

**Probable cause**

You have tried to update a column in a query that is a database expression rather than a column in a table.

**CHECKPOINT command requires a rollback log**

Item	Value
SQLCode	-212
Constant	SQLE_CHECKPOINT_REQUIRES_UNDO
SQLState	42W20
ODBC State	40001

**Probable cause**

You cannot use a CHECKPOINT command when the database engine is running in bulk mode without a rollback log.

## Client/server communications protocol mismatch

Item	Value
SQLCode	-88
Constant	SQLE_PROTOCOL_MISMATCH
SQLState	08W15
ODBC State	08S01

**Probable cause** The multi-user client was unable to start because the protocol versions of the client and the running server do not match. Make sure the client and server software are the same version.

## Column %1 in foreign key has a different definition than primary key

Item	Value
SQLCode	-113
Constant	SQLE_INVALID_FOREIGN_KEY_DEF
SQLState	53030
ODBC State	23000
Parameter 1	Name of the problem column.

**Probable cause** The data type of the column in the foreign key is not the same as the data type of the column in the primary key. Change the definition of one of the columns using ALTER TABLE.

## Column '%1' found in more than one table -- need a correlation name

Item	Value
SQLCode	-144
Constant	SQLE_COLUMN_AMBIGUOUS
SQLState	52002
ODBC State	SJS01



Item	Value
Parameter 1	Name of the ambiguous column.

**Probable cause**

You have not put a correlation name on a column that is found in more than one of the tables referenced in a query. You need to add a correlation name to the reference.

**Column '%1' in table '%2' cannot be NULL**

Item	Value
SQLCode	-195
Constant	SQLE_COLUMN_CANNOT_BE_NULL
SQLState	23502
ODBC State	23000
Parameter 1	Name of the column that cannot be NULL.
Parameter 2	Name of the table containing the column.

**Probable cause**

You have not supplied a value where a value is required. The column definition prohibits NULL values or the column is part of a NOT NULL foreign key.

**Column '%1' not found**

Item	Value
SQLCode	-143
Constant	SQLE_COLUMN_NOT_FOUND
SQLState	52003
ODBC State	S0022
Parameter 1	Name of the column that could not be found.

**Probable cause**

You have misspelled the name of a column, or the column you are looking for is in a different table.

## **COMMIT/ROLLBACK not allowed within atomic operation**

<b>Item</b>	<b>Value</b>
SQLCode	-267
Constant	SQLE_ATOMIC_OPERATION
SQLState	42W28
ODBC State	37000

**Probable cause** A COMMIT or ROLLBACK statement was encountered while executing within an atomic operation.

## **COMMIT/ROLLBACK not allowed within trigger actions**

<b>Item</b>	<b>Value</b>
SQLCode	-273
Constant	SQLE_INVALID_TRIGGER_STATEMENT
SQLState	2D501
ODBC State	37000

**Probable cause** An attempt was made to execute a statement that is not allowed while performing a trigger action. COMMIT and ROLLBACK statements cannot be executed from a trigger.

## **Communication buffer underflow**

<b>Item</b>	<b>Value</b>
SQLCode	-73
Constant	SQLE_COMMUNICATIONS_UNDERFLOW
SQLState	08W31
ODBC State	08S01

**Probable cause** The multi-user client or server has received a network buffer containing insufficient data. If this error occurs, it should be reported to Watcom.

## Communication error

Item	Value
SQLCode	-85
Constant	SQLE_COMMUNICATIONS_ERROR
SQLState	08W12
ODBC State	08S01

**Probable cause** There is a communication problem between the multi-user client and server. This happens most frequently when the multi-user client was unable to start because a communication error occurred while trying to locate the server.

## Connection not found

Item	Value
SQLCode	-108
Constant	SQLE_CONNECTION_NOT_FOUND
SQLState	08W02
ODBC State	08003

**Probable cause** The specified connection name on a DISCONNECT or SET CONNECTION statement is invalid.

## Connection was terminated

Item	Value
SQLCode	-308
Constant	SQLE_CONNECTION_TERMINATED
SQLState	40W07
ODBC State	S1000

**Probable cause** Your database connection has been terminated by a DBA executing a DROP CONNECTION command. Your transaction was rolled back.

## Connections to database have been disabled

Item	Value
SQLCode	-99
Constant	SQLE_CONNECTIONS_DISABLED
SQLState	08W04
ODBC State	08005

### Probable cause

Connections to the multi-user server have been disabled on the server console. You will receive this error until they have been reenabled on the server console.

## Correlation name '%1' not found

Item	Value
SQLCode	-142
Constant	SQLE_CORRELATION_NAME_NOT_FOUND
SQLState	52W02
ODBC State	S0002
Parameter 1	Name of the invalid correlation name.

### Probable cause

You have misspelled a correlation name, or you have used a table name instead of the correlation name.

## Could not allocate resources to call external function

Item	Value
SQLCode	-622
Constant	SQLE_ERROR_CALLING_FUNCTION
SQLState	WW008
ODBC State	S1000

**Probable cause** The external function could not be called due to a shortage of operating system resources. If the operating system supports threads, the maximum thread count should be increased.

### Could not find the named function in the dynamic library

Item	Value
SQLCode	-621
Constant	SQLE_COULD_NOT_FIND_FUNCTION
SQLState	WW007
ODBC State	S1000

**Probable cause** The external function could not be found in the dynamic library.

### Could not load the dynamic library

Item	Value
SQLCode	-620
Constant	SQLE_COULD_NOT_LOAD_LIBRARY
SQLState	WW006
ODBC State	S1000

**Probable cause** The library named in an external function call could not be loaded.

### Cursor already open

Item	Value
SQLCode	-172
Constant	SQLE_CURSOR_ALREADY_OPEN
SQLState	24502
ODBC State	24000

**Probable cause** You attempted to OPEN a cursor that is already open.

## Cursor has not been declared

Item	Value
SQLCode	-170
Constant	SQLE_CURSOR_NOT_DECLARED
SQLState	24W01
ODBC State	24000

**Probable cause** You attempted to OPEN a cursor that has not been declared.

## Cursor not open

Item	Value
SQLCode	-180
Constant	SQLE_CURSOR_NOT_OPEN
SQLState	24501
ODBC State	34000

**Probable cause** You attempted to OPEN a cursor that has not been declared.

## Cursor operation conflict

Item	Value
SQLCode	108
Constant	SQLE_CURSOR_OPERATION_CONFLICT
SQLState	01001
ODBC State	(handled by ODBC driver)

**Probable cause** You have attempted to perform an operation on the current row of a cursor, but the row has been modified by a searched UPDATE or DELETE or by a positioned UPDATE or DELETE.

## Data definition statements not allowed in procedures or triggers

Item	Value
SQLCode	-623
Constant	SQLE_DDL_NOT_ALLOWED_IN_PROCEDURES
SQLState	52W21
ODBC State	S1000

### Probable cause

The procedure or trigger definition contains a data definition statement (such as CREATE, DROP, GRANT, REVOKE, ALTER). These statements are not allowed within procedures or triggers.

## Database backup not started

Item	Value
SQLCode	-241
Constant	SQLE_BACKUP_NOT_STARTED
SQLState	WB002
ODBC State	S1000

### Probable cause

A database backup could not be started. Either you do not have DBA authority, or another backup has started and not completed.

## Database engine already running

Item	Value
SQLCode	-96
Constant	SQLE_ENGINE_ALREADY_RUNNING
SQLState	08W23
ODBC State	S1000

### Probable cause

The database engine was not able to start on a db\_start\_engine call because it was already running.

## Database engine not running

Item	Value
SQLCode	-100
Constant	SQLE_ENGINE_NOT_RUNNING
SQLState	08W01
ODBC State	08001

**Probable cause** You have not run the database engine or network requestor or the interface library is unable to find it.

## Database engine not running in multi-user mode

Item	Value
SQLCode	-89
Constant	SQLE_ENGINE_NOT_MULTUSER
SQLState	08W16
ODBC State	08001

**Probable cause** The database was started for bulk loading (the -b switch) and cannot be used as a multi-user engine. Stop the database, and start again without -b. In the DOS version of Watcom SQL 3.0, the database engine was not started in multi-user mode.

## Database name not unique

Item	Value
SQLCode	-77
Constant	SQLE_ALIAS_CLASH
SQLState	08W27
ODBC State	08001

**Probable cause** The database cannot be loaded as its name is conflicting with a previously loaded database.



## Database name required to start engine

Item	Value
SQLCode	-87
Constant	SQLE_DATABASE_NAME_REQUIRED
SQLState	08W14
ODBC State	08001

### Probable cause

A database name is required to start the database engine or the multi-user client, but it was not specified.

## Database option '%1' for user '%2' has an invalid setting

Item	Value
SQLCode	113
Constant	SQLE_INVALID_OPTION_ON_CONNECT
SQLState	01W11
ODBC State	(handled by ODBC driver)
Parameter 1	Name of the database option that has the invalid value.
Parameter 2	Name of the user attempting to connect.

### Probable cause

Upon processing a connection request for a specific user, the engine processed a database option from the SYSOPTIONS table which had an invalid setting. The erroneous option setting is ignored; in its place, the engine will use the default option value for the current database.

## Database was initialized with an older version of the software

Item	Value
SQLCode	-266
Constant	SQLE_OLD_DBINIT
SQLState	42W27
ODBC State	37000

**Probable cause**

The database is missing some system table definitions required for this statement. These system table definitions are normally created when a database is initialized. The database should be unloaded and reloaded into a database that has been initialized with a newer version of SQL Anywhere or use DBUPGRADE to upgrade the database to the most recent version.

## Database's page size too big

Item	Value
SQLCode	-97
Constant	SQLE_PAGE_SIZE_TOO_BIG
SQLState	08W22
ODBC State	08004

**Probable cause**

You have attempted to start a database with a page size that exceeds the maximum page size of the running engine. Restart the engine with this database named on the command line.

## Dblib/database engine version mismatch

Item	Value
SQLCode	-231
Constant	SQLE_DBLIB_ENGINE_MISMATCH
SQLState	08W19
ODBC State	08001

**Probable cause**

Your executable uses a database interface library that does not match the version number of the database engine.

## DbSPACE '%1' not found

Item	Value
SQLCode	-138
Constant	SQLE_DBSPACE_NOT_FOUND

Item	Value
SQLState	52W13
ODBC State	S0002
Parameter 1	Name of the dbspace that could not be found.

**Probable cause** The named dbspace was not found.

## Deadlock detected

Item	Value
SQLCode	-306
Constant	SQLE_DEADLOCK
SQLState	40001
ODBC State	40001

**Probable cause** You have attempted to read or write a row and it is locked by another user. Also, the other user is blocked directly or indirectly on your own transaction. This is a deadlock situation and your transaction has been chosen as the one to rollback.

## Disallowed language extension detected in syntax near '%1'

Item	Value
SQLCode	-627
Constant	SQLE_INVALID_SYNTAX_EXTENSION
SQLState	0AW03
ODBC State	S1000
Parameter 1	The word or symbol where the syntax has been detected.

**Probable cause** The command you are trying to execute contains extensions to ANSI 1992 Entry SQL that are not allowed by the current settings.

## Disk full '%1' -- transaction rolled back

Item	Value
SQLCode	-304
Constant	SQLE_DEVICE_FULL
SQLState	40W03
ODBC State	S1000

**Probable cause** Your hard disk is out of free space. A ROLLBACK WORK command has been automatically executed.

## Division by zero

Item	Value
SQLCode	-628
Constant	SQLE_DIV_ZERO_ERROR
SQLState	22012
ODBC State	22012

**Probable cause** A division by zero operation was detected.

## Do not have permission to %1

Item	Value
SQLCode	-121
Constant	SQLE_PERMISSION_DENIED
SQLState	42501
ODBC State	42000
Parameter 1	Description of the type of permission lacking.

**Probable cause** You do not have the required permission to do the specified action. You have not been granted permission to use a table belonging to another userid.

## Duplicate insert column

Item	Value
SQLCode	-637
Constant	SQLE_DUPLICATE_INSERT_COLUMN
SQLState	42W41
ODBC State	42000

**Probable cause** A duplicate column name was used in the list of insert columns.

## Duplicate referencing column

Item	Value
SQLCode	-636
Constant	SQLE_DUPLICATE_REFERENCING_COLUMN
SQLState	42W40
ODBC State	42000

**Probable cause** A duplicate column name was used in the list of referencing columns.

## Dynamic memory exhausted!

Item	Value
SQLCode	-78
Constant	SQLE_DYNAMIC_MEMORY_EXHAUSTED
SQLState	08W26
ODBC State	S1001

**Probable cause** A failure occurred when trying to allocate dynamic memory.

## Error in assignment

Item	Value
SQLCode	-641
Constant	SQLE_ERROR_IN_ASSIGNMENT
SQLState	22005
ODBC State	22005

### Probable cause

In a GET DESCRIPTOR statement, the data type of the host variable must correspond to the data type of the descriptor item.

## Error inserting into cursor

Item	Value
SQLCode	-184
Constant	SQLE_PUT_CURSOR_ERROR
SQLState	09W01
ODBC State	S1000

### Probable cause

An error has occurred while inserting into a cursor.

## Error number %1 for RAISERROR is less than 17000

Item	Value
SQLCode	-296
Constant	SQLE_ERROR_NUMBER_OUT_OF_RANGE
SQLState	53W07
ODBC State	37000
Parameter 1	Error number.

### Probable cause

The error number used in a RAISERROR statement is invalid. The number must be greater than or equal to 17000.

## Error opening cursor

Item	Value
SQLCode	-171
Constant	SQLE_OPEN_CURSOR_ERROR
SQLState	07003
ODBC State	24000

**Probable cause** You have attempted to open a cursor on a statement that is not a SELECT statement or a CALL statement.

## Error writing to log file

Item	Value
SQLCode	-107
Constant	SQLE_ERROR_WRITING_LOG
SQLState	08W17
ODBC State	S1000

**Probable cause** The database engine got an I/O error writing the log file. Perhaps the disk is full or the log file name is invalid.

## Feature '%1' not implemented

Item	Value
SQLCode	-134
Constant	SQLE_NOT_IMPLEMENTED
SQLState	0A000
ODBC State	S1000
Parameter 1	The unimplemented feature.

**Probable cause** The requested operation or feature is not implemented in SQL Anywhere.

## Foreign key '%1' for table '%2' duplicates an existing foreign key

Item	Value
SQLCode	-251
Constant	SQLE_DUPLICATE_FOREIGN_KEY
SQLState	52W06
ODBC State	23000
Parameter 1	The role name of the new foreign key.
Parameter 2	The table containing the foreign key.

**Probable cause** You have attempted to define a foreign key that already exists.

## Foreign key name '%1' not found

Item	Value
SQLCode	-145
Constant	SQLE_FOREIGN_KEY_NAME_NOT_FOUND
SQLState	52W07
ODBC State	37000
Parameter 1	Name of the non-existing foreign key.

**Probable cause** You have misspelled the name of a foreign key or the foreign key does not exist.

## Format string argument number %1 is invalid

Item	Value
SQLCode	-294
Constant	SQLE_INVALID_FORMAT_STRING_ARG_NUM
SQLState	53W08
ODBC State	37000
Parameter 1	Argument number.



**Probable cause**

An argument number in the format string for a PRINT or RAISERROR statement is invalid. The number must be between 1 and 20 and must not exceed the number of arguments provided.

## Function or column reference to '%1' in the select list must also appear in a GROUP BY

Item	Value
SQLCode	-149
Constant	SQLE_INVALID_GROUP_SELECT
SQLState	53003
ODBC State	37000
Parameter 1	Name of the column referenced directly, or in an expression, that must be in the GROUP BY clause.

**Probable cause**

In a query using GROUP BY, select list items that are not aggregate functions must also appear in the GROUP BY clause. If the select list item is a column reference or an alias, simply add the column name or alias to the GROUP BY clause. If the select list item is a scalar function, ensure that the function's arguments in the GROUP BY clause match exactly with those in the select list. In some cases, you may want to use the MAX function on the column name (or another aggregate function) instead of adding the column to the GROUP BY clause.

## GRANT of column permission on view not allowed

Item	Value
SQLCode	-635
Constant	SQLE_NO_COLUMN_PERMS_FOR_VIEWS
SQLState	52W22
ODBC State	S1000

**Probable cause**

Permissions on columns cannot be granted for views.

## I/O error %1 -- transaction rolled back

Item	Value
SQLCode	-305
Constant	SQLE_DEVICE_ERROR
SQLState	40W04
ODBC State	S1000

**Probable cause** SQL Anywhere has detected a problem with your hard disk. If you cannot find a hardware error using the operating system disk check utility (eg. in DOS, chkdsk, and in QNX, chkfsys), report the problem to SQL Anywhere technical support. A ROLLBACK WORK command has been automatically executed.

## Identifier '%1' too long

Item	Value
SQLCode	-250
Constant	SQLE_IDENTIFIER_TOO_LONG
SQLState	54003
ODBC State	37000
Parameter 1	The identifier in error.

**Probable cause** An identifier is longer than 128 characters.

## Incomplete transactions prevent transaction log renaming

Item	Value
SQLCode	-242
Constant	SQLE_BACKUP_CANNOT_RENAME_LOG_YET
SQLState	WB003
ODBC State	S1000

**Probable cause**

The last page in the transaction log was read by a call to db\_backup. One or more currently active connections have partially completed transactions, preventing the transaction log file from being renamed. The db\_backup call should be reissued.

**Index '%1' for table '%2' would not be unique**

Item	Value
SQLCode	-196
Constant	SQLE_INDEX_NOT_UNIQUE
SQLState	23505
ODBC State	23000
Parameter 1	Name of the index that would not be unique.
Parameter 2	Name of the table that contains the index.

**Probable cause**

You have inserted or updated a row that has the same value as another row in some column, and there is a constraint that does not allow two rows to have the same value in that column.

**Index name '%1' not unique**

Item	Value
SQLCode	-111
Constant	SQLE_INDEX_NAME_NOT_UNIQUE
SQLState	52W04
ODBC State	S0011
Parameter 1	Name of the invalid index.

**Probable cause**

You have attempted to create an index with a name of an existing index.

## INSERT/DELETE on cursor can modify only one table

Item	Value
SQLCode	-199
Constant	SQLE_ONLY_ONE_TABLE
SQLState	09W04
ODBC State	37000

**Probable cause** You have attempted to INSERT into a cursor and have specified values for more than one table; or you have tried to DELETE from a cursor that involves a join. INSERT into one table at a time. For DELETE, use the FROM clause to specify which table you wish to delete from.

## Integrated logon failed

Item	Value
SQLCode	-245
Constant	SQLE_INTEGRATED_LOGON_FAILED
SQLState	28W04
ODBC State	28000

**Probable cause** The integrated logon failed, possibly because if the user doesn't have a system account on the server machine, or for other reasons.

## Integrated logons are not permitted

Item	Value
SQLCode	-205
Constant	SQLE_INVALID_STANDARD_LOGON
SQLState	28W02
ODBC State	28000

**Probable cause** The engine logon type switch (gl) is set to STANDARD logon type, and the user has attempted an integrated logon.

## Internal database error %1 -- transaction rolled back

Item	Value
SQLCode	-301
Constant	SQLE_DATABASE_ERROR
SQLState	40W01
ODBC State	S1000
Parameter 1	Identification of the error.

**Probable cause** This error indicates an internal database error, and should be reported to SQL Anywhere technical support. A ROLLBACK WORK command has been automatically executed.

## Invalid absolute or relative offset in FETCH

Item	Value
SQLCode	-263
Constant	SQLE_INVALID_FETCH_POSITION
SQLState	42W25
ODBC State	37000

**Probable cause** The offset specified in a FETCH was invalid or NULL.

## Invalid column number

Item	Value
SQLCode	-159
Constant	SQLE_INVALID_COLUMN_NUMBER
SQLState	42W13
ODBC State	S1000

**Probable cause** The column number in a GET DATA command is invalid.

## Invalid data conversion

Item	Value
SQLCode	103
Constant	SQLE_CANNOT_CONVERT
SQLState	01W03
ODBC State	07006

### Probable cause

The database could not convert a value to the required type. This is either a value supplied to the database on an insert, update or as a host bind variable, or a value retrieved from the database into a host variable or SQLDA.

## Invalid database engine command line

Item	Value
SQLCode	-81
Constant	SQLE_INVALID_COMMAND_LINE
SQLState	08W08
ODBC State	08001

### Probable cause

It was not possible to start the database engine or multi-user client because the command line was invalid. of how the engine is started.

## Invalid datatype for column in WRITETEXT or READTEXT

Item	Value
SQLCode	-609
Constant	SQLE_INVALID_TEXT_IMAGE_DATATYPE
SQLState	53W09
ODBC State	S1000

### Probable cause

The column referenced in a WRITETEXT or READTEXT statement is not defined for storing text or image data.

## Invalid descriptor index

Item	Value
SQLCode	-640
Constant	SQLE_INVALID_DESCRIPTOR_INDEX
SQLState	07009
ODBC State	07009

**Probable cause** The index number used with respect to a descriptor area is out of range.

## Invalid escape character '%1'

Item	Value
SQLCode	-629
Constant	SQLE_INVALID_ESCAPE_CHAR
SQLState	22019
ODBC State	22019

**Probable cause** Escape character string length must be exactly one.

## Invalid escape sequence '%1'

Item	Value
SQLCode	-630
Constant	SQLE_INVALID_ESCAPE_SEQ
SQLState	22025
ODBC State	22025

**Probable cause** LIKE pattern contains an invalid use of the escape character. The escape character may only precede the special characters '%', '\_', '[', and the escape character itself.

## Invalid expression near '%1'

Item	Value
SQLCode	-156
Constant	SQLE_EXPRESSION_ERROR
SQLState	42W08
ODBC State	37000
Parameter 1	The invalid expression.

**Probable cause** You have an expression which the database engine cannot understand. For example, you might have tried to add two dates.

## Invalid host variable

Item	Value
SQLCode	-155
Constant	SQLE_VARIABLE_INVALID
SQLState	42W07
ODBC State	37000

**Probable cause** A host variable supplied to the database using the C language interface as either a host variable or through an SQLDA is invalid.

## Invalid local database switch

Item	Value
SQLCode	-79
Constant	SQLE_INVALID_LOCAL_OPTION
SQLState	08W25
ODBC State	08001

**Probable cause** An invalid local database switch was found in the DBS option.



## Invalid operation for this cursor

Item	Value
SQLCode	-187
Constant	SQLE_CURSOROP_NOT_ALLOWED
SQLState	09W02
ODBC State	24000

**Probable cause** An operation that is not allowed was attempted on a cursor.

## Invalid option '%1' -- no PUBLIC setting exists

Item	Value
SQLCode	-200
Constant	SQLE_INVALID_OPTION
SQLState	42W16
ODBC State	37000
Parameter 1	Name of the invalid option.

**Probable cause** You have probably misspelled the name of an option in the SET OPTION command. You can only define an option for a user if the database administrator has supplied a PUBLIC value for that option.

## Invalid parameter

Item	Value
SQLCode	-95
Constant	SQLE_INVALID_PARAMETER
SQLState	08W24
ODBC State	08004

**Probable cause** An error occurred while parsing the string parameter associated with one of the entry points: db\_start\_engine(), db\_start\_database(), db\_stop\_engine(), db\_stop\_database(), db\_string\_connect().

## Invalid prepared statement type

Item	Value
SQLCode	-133
Constant	SQLE_INVALID_STATEMENT_TYPE
SQLState	07W03
ODBC State	S1000

**Probable cause** This is an internal C language interface error. If it occurs, it should be reported to SQL Anywhere technical support.

## Invalid REFERENCES clause in trigger definition

Item	Value
SQLCode	-272
Constant	SQLE_INVALID_TRIGGER_COL_REFS
SQLState	52W12
ODBC State	37000

**Probable cause** The REFERENCES clause in a trigger definition is invalid. An OLD correlation name may have been specified in a BEFORE INSERT trigger, or a NEW correlation name may have been specified in an AFTER DELETE trigger. In both cases, the values do not exist and cannot be referenced.

## Invalid setting for option '%1'

Item	Value
SQLCode	-201
Constant	SQLE_INVALID_OPTION_SETTING
SQLState	42W17
ODBC State	37000
Parameter 1	Name of the invalid option.

**Probable cause**

You have supplied an invalid value for an option in the SET command. Some options only allow numeric values, while other options only allow the values on and off:evaluate..

**Invalid SQL Anywhere HLI callback function**

Item	Value
SQLCode	-405
Constant	SQLE_HLI_BAD_CALLBACK
SQLState	42W12
ODBC State	42W12

**Probable cause**

WSQL HLI needed to use a callback function, but the function has not been registered using the wsqlregisterfuncs entry point.

**Invalid SQL Anywhere HLI command syntax**

Item	Value
SQLCode	-400
Constant	SQLE_HLI_BAD_SYNTAX
SQLState	42W11
ODBC State	42W11

**Probable cause**

The command string that you sent to wsqlexec cannot be understood. Make sure that all of the keywords in the command string are spelled properly, and that variable names (such as host variable, cursor or statement names) are not too long.

**Invalid SQL Anywhere HLI cursor name**

Item	Value
SQLCode	-401
Constant	SQLE_HLI_BAD_CURSOR

<b>Item</b>	<b>Value</b>
SQLState	34W01
ODBC State	34W01

**Probable cause** The cursor name indicated in your command is not a valid one. For instance, this error would occur if you tried to close a cursor that had never even been declared.

### **Invalid SQL Anywhere HLI host variable name**

<b>Item</b>	<b>Value</b>
SQLCode	-403
Constant	SQLE_HLI_BAD_HOST_VAR_NAME
SQLState	42W09
ODBC State	42W09

**Probable cause** You have used a host variable, and the host variable callback function does not recognize it.

### **Invalid SQL Anywhere HLI host variable value**

<b>Item</b>	<b>Value</b>
SQLCode	-404
Constant	SQLE_HLI_BAD_HOST_VAR_VALUE
SQLState	42W10
ODBC State	42W10

**Probable cause** You have used a host variable, and the host variable value is too long.

### **Invalid SQL Anywhere HLI statement name**

<b>Item</b>	<b>Value</b>
SQLCode	-402

Item	Value
Constant	SQLE_HLI_BAD_STATEMENT
SQLState	26W01
ODBC State	26W01

**Probable cause** The statement name indicated in your command is not a valid one. This typically indicates that you have failed to prepare the statement.

### Invalid SQL descriptor name

Item	Value
SQLCode	-642
Constant	SQLE_INVALID_DESCRIPTOR_NAME
SQLState	33000
ODBC State	33000

**Probable cause** You have attempted to deallocate a descriptor which has not been allocated.

### Invalid statement

Item	Value
SQLCode	-130
Constant	SQLE_INVALID_STATEMENT
SQLState	07W02
ODBC State	S1000

**Probable cause** The statement identifier (generated by PREPARE) passed to the database for a further operation is invalid.

### Invalid TEXTPTR value used with WRITETEXT or READTEXT

Item	Value
SQLCode	-608

<b>Item</b>	<b>Value</b>
Constant	SQLE_INVALID_TEXTPTR_VALUE
SQLState	22W03
ODBC State	S1000

**Probable cause** The value supplied as the TEXTPTR for a WRITETEXT or READTEXT statement is invalid.

## **Invalid type on DESCRIBE statement**

<b>Item</b>	<b>Value</b>
SQLCode	-161
Constant	SQLE_INVALID_DESCRIBE_TYPE
SQLState	07W01
ODBC State	(handled by ODBC driver)

**Probable cause** This is an internal C language interface error. If it occurs, it should be reported to SQL Anywhere technical support.

## **Invalid userid and password on preprocessed module**

<b>Item</b>	<b>Value</b>
SQLCode	-104
Constant	SQLE_INVALID_MODULE_LOGON
SQLState	28W01
ODBC State	28000

**Probable cause** A userid and password were specified when a module was preprocessed but the userid or password is invalid.

## Invalid userid or password

Item	Value
SQLCode	-103
Constant	SQLE_INVALID_LOGON
SQLState	28000
ODBC State	28000

### Probable cause

The user has supplied an invalid userid or an incorrect password. ISQL will handle this error by presenting a connection dialog to the user.

## Invalid value for column '%1' in table '%2'

Item	Value
SQLCode	-209
Constant	SQLE_INVALID_COLUMN_VALUE
SQLState	23506
ODBC State	23000
Parameter 1	Name of the column that was assigned an invalid value.
Parameter 2	Name of the table containing the column.

### Probable cause

An INSERT or UPDATE has specified a value for a column that violates a CHECK constraint, and the INSERT or UPDATE were not done because of the error. Note that a CHECK constraint is violated if it evaluates to FALSE; it is okay if it evaluates to TRUE or UNKNOWN.

## Item '%1' already exists

Item	Value
SQLCode	-110
Constant	SQLE_NAME_NOT_UNIQUE
SQLState	52010
ODBC State	S0001

Item	Value
Parameter 1	Name of the item that already exists.

**Probable cause** You have tried to create a table, view, column, foreign key, or publication with the same name as an existing one.

## Label '%1' not found

Item	Value
SQLCode	-262
Constant	SQLE_LABEL_NOT_FOUND
SQLState	42W24
ODBC State	37000
Parameter 1	Name of the label that could not be found.

**Probable cause** The label referenced in a LEAVE statement was not found.

## Language extension

Item	Value
SQLCode	-135
Constant	SQLE_LANGUAGE_EXTENSION
SQLState	0AW01
ODBC State	S1000

**Probable cause** The requested operation is valid in some versions of SQL, but not in SQL Anywhere.

## Language extension detected in syntax

Item	Value
SQLCode	107
Constant	SQLE_SYNTAX_EXTENSION_WARNING



Item	Value
SQLState	01W07
ODBC State	(handled by ODBC driver)

**Probable cause** The command you are executing contains extensions to ANSI 1992 Entry SQL.

## Mismatch between external function platform specifier and current operating system

Item	Value
SQLCode	-618
Constant	SQLE_EXTERNAL_PLATFORM_FAILURE
SQLState	WW004
ODBC State	S1000

**Probable cause** A call to an external entry point in a dynamically loaded module was qualified by an operating system which was not the operating system on which the engine/server is currently executing.

## More columns are being dropped from table %1 than are defined

Item	Value
SQLCode	-124
Constant	SQLE_TOO_MANY_COLUMNS_DELETED
SQLState	42W44
ODBC State	42000

**Probable cause** The number of columns in a table can never fall below one. Your ALTER TABLE has more drop/delete column clauses than the current number of columns in the table.

## More info required

Item	Value
SQLCode	112
Constant	SQLE_MORE_INFO
SQLState	01W09
ODBC State	(handled by ODBC driver)

**Probable cause** More information is required to complete the request. This is used internally in the database interface library to process a unified logon. It should not be returned to an application.

## More than one table is identified as '%1'

Item	Value
SQLCode	-139
Constant	SQLE_CORRELATION_NAME_AMBIGUOUS
SQLState	52012
ODBC State	SG001
Parameter 1	Ambiguous correlation name.

**Probable cause** You have identified two tables in the same FROM clause with the same correlation name.

## Need a dynamic library name

Item	Value
SQLCode	-619
Constant	SQLE_REQUIRE_DLL_NAME
SQLState	WW005
ODBC State	S1000

**Probable cause** The name of the external function to call did not contain a library name specifier.

## No current row of cursor

Item	Value
SQLCode	-197
Constant	SQLE_NO_CURRENT_ROW
SQLState	24503
ODBC State	24000

### Probable cause

You have attempted to perform an operation on the current row of a cursor, but there is no current row. The cursor is before the first row of the cursor, after the last row or is on a row that has since been deleted.

## No data

Item	Value
SQLCode	100
Constant	SQLE_NO_DESCRIPTOR_DATA
SQLState	02W01
ODBC State	02W01

### Probable cause

You have positioned a cursor beyond the beginning or past the end of the query. There is no row at that position. You have attempted to access a descriptor area using an index value larger than the number of variables in the descriptor.

## No indicator variable provided for NULL result

Item	Value
SQLCode	-181
Constant	SQLE_NO_INDICATOR
SQLState	22002
ODBC State	S1000

**Probable cause** You tried to retrieve a value from the database that was the NULL value but you did not provide an indicator variable for that value.

### No primary key value for foreign key '%1' in table '%2'

Item	Value
SQLCode	-194
Constant	SQLE_INVALID_FOREIGN_KEY
SQLState	23503
ODBC State	23000
Parameter 1	Name of the foreign key.
Parameter 2	Name of the table with the foreign key.

**Probable cause** You have tried to insert or update a row that has a foreign key for another table, and the value for the foreign key is not NULL and there is not a corresponding value in the primary key

### Not allowed while %1 is using the database

Item	Value
SQLCode	-211
Constant	SQLE_MUST_BE_ONLY_CONNECTION
SQLState	42W19
ODBC State	40001

**Probable cause** You have attempted a data definition language operation that cannot be completed while another is connected.

### Not connected to SQL database

Item	Value
SQLCode	-101
Constant	SQLE_NOT_CONNECTED

Item	Value
SQLState	08003
ODBC State	08003

**Probable cause** You have not connected to the database, or you have executed the DISCONNECT command and have not connected to the database again.

## Not enough fields allocated in SQLDA

Item	Value
SQLCode	-182
Constant	SQLE_SQLDA_TOO_SMALL
SQLState	07002
ODBC State	07001

**Probable cause** There are not enough fields in the SQLDA to retrieve all of the values requested.

## Not enough memory to start

Item	Value
SQLCode	-86
Constant	SQLE_NO_MEMORY
SQLState	08W13
ODBC State	S1001

**Probable cause** The database engine or multi-user client executable was loaded but was unable to start because there is not enough memory to run properly.

## Not enough values for host variables

Item	Value
SQLCode	-188

<b>Item</b>	<b>Value</b>
Constant	SQLE_NOT_ENOUGH_HOST_VARS
SQLState	07001
ODBC State	07001

**Probable cause** You have not provided enough host variables for either the number of bind variables, or the command, or the number of select list items.

### **Null value eliminated in aggregate function**

<b>Item</b>	<b>Value</b>
SQLCode	109
Constant	SQLE_NULL_VALUE_ELIMINATED
SQLState	01003
ODBC State	(handled by ODBC driver)

**Probable cause** Value of the expression argument of the aggregate function evaluated to NULL for one or more rows.

### **Number in ORDER BY is too large**

<b>Item</b>	<b>Value</b>
SQLCode	-152
Constant	SQLE_INVALID_ORDER
SQLState	53005
ODBC State	37000

**Probable cause** You have used an integer in an ORDER BY list and the integer is larger than the number of columns in the select list.

## Number of columns does not match SELECT

Item	Value
SQLCode	-114
Constant	SQLE_VIEW_DEFINITION_ERROR
SQLState	53011
ODBC State	21S01

**Probable cause** An INSERT command contains a SELECT with a different number of columns than the INSERT.

## Only PUBLIC settings are allowed for option '%1'

Item	Value
SQLCode	-202
Constant	SQLE_NOT_PUBLIC_ID
SQLState	42W43
ODBC State	37000
Parameter 1	Name of the option.

**Probable cause** The option specified in the SET OPTION command is PUBLIC only. You cannot define this option for any other user.

## Only the DBA can set the option %1

Item	Value
SQLCode	-204
Constant	SQLE_OPTION_REQUIRES_DBA
SQLState	42W46
ODBC State	37000

**Probable cause** The option specified in the SET OPTION command can only be set by a user having DBA authority.

## Operation would cause a group cycle

Item	Value
SQLCode	-122
Constant	SQLE_GROUP_CYCLE
SQLState	42W02
ODBC State	37000

**Probable cause** You have tried to add a member to group that would result in a member belonging to itself (perhaps indirectly).

## Parameter '%1' not found in procedure '%2'

Item	Value
SQLCode	-615
Constant	SQLE_INVALID_PARAMETER_NAME
SQLState	42W47
ODBC State	37000

**Probable cause** The procedure parameter name does not match a parameter for this procedure. Check the spelling of the parameter name.

## Parameter name missing in call to procedure '%1'

Item	Value
SQLCode	-639
Constant	SQLE_PARAMETER_NAME_MISSING
SQLState	42W42
ODBC State	37000
Parameter 1	The name of the procedure

**Probable cause** Positional arguments were specified after keyword arguments in a call to this procedure.



## Passthrough statement inconsistent with current passthrough

Item	Value
SQLCode	-287
Constant	SQLE_PASSTHROUGH_INCONSISTENT
SQLState	5RW08
ODBC State	S0002

### Probable cause

Passthrough is additive, in that subsequent passthrough statements add to the list of users receiving passthrough. The passthrough statements must all be PASSTHROUGH ONLY or none should be PASSTHROUGH ONLY.

## Primary key column '%1' already defined

Item	Value
SQLCode	-119
Constant	SQLE_PRIMARY_KEY_COLUMN_DEFINED
SQLState	52009
ODBC State	23000
Parameter 1	Name of the column that is already in the primary key.

### Probable cause

You have listed the same column name twice in the definition of a primary key.

## Primary key for row in table '%1' is referenced in another table

Item	Value
SQLCode	-198
Constant	SQLE_PRIMARY_KEY_VALUE_REF
SQLState	23W05
ODBC State	23000
Parameter 1	The name of the table with a primary key that is referenced.

**Probable cause** You have attempted to delete or modify a primary key that is referenced elsewhere in the database.

### Primary key for table '%1' is not unique

Item	Value
SQLCode	-193
Constant	SQLE_PRIMARY_KEY_NOT_UNIQUE
SQLState	23W01
ODBC State	23000
Parameter 1	Name of the table where the problem was detected.

**Probable cause** You have tried to add a new row to a table where the new row has the same primary key as an existing row. The database has not added the incorrect row to the database. For example, you might have added a student with student number 86004 and there is already a row for a student with that number.

### Procedure '%1' not found

Item	Value
SQLCode	-265
Constant	SQLE_PROCEDURE_NOT_FOUND
SQLState	52W09
ODBC State	S0002
Parameter 1	Name of the procedure that could not be found.

**Probable cause** You have misspelled the name of a procedure, or you have connected with a different userid and forgotten to qualify a procedure name with a user name.

## Procedure has completed

Item	Value
SQLCode	105
Constant	SQLE_PROCEDURE_COMPLETE
SQLState	01W05
ODBC State	(handled by ODBC driver)

**Probable cause** An OPEN or a RESUME has caused a procedure to execute to completion. There are no more result sets available from this procedure. This warning will also be returned if you attempt to RESUME a cursor on a SELECT statement.

## Procedure in use

Item	Value
SQLCode	-215
Constant	SQLE_PROCEDURE_IN_USE
SQLState	42W23
ODBC State	40001

**Probable cause** You have attempted to DROP a procedure that is being used by other active users of the database.

## Procedure or trigger calls have nested too deeply

Item	Value
SQLCode	-274
Constant	SQLE_NESTING_TOO_DEEP
SQLState	42W29
ODBC State	37000

**Probable cause** You have probably defined a procedure or trigger that causes unlimited recursion.

## Publication '%1' not found

Item	Value
SQLCode	-280
Constant	SQLE_PUBLICATION_NOT_FOUND
SQLState	5RW01
ODBC State	S0002
Parameter 1	Name of the publication that could not be found.

**Probable cause** You have misspelled the name of a publication, or you have connected with a different userid and forgotten to qualify a publication name with a user name.

## RAISERROR executed: %1

Item	Value
SQLCode	-631
Constant	SQLE_RAISERROR_STMT
SQLState	WW012
ODBC State	S1000
Parameter 1	RAISERROR message string

**Probable cause** A RAISERROR statement has been executed.

## Remote message type '%1' not found

Item	Value
SQLCode	-286
Constant	SQLE_NOT_REMOTE_TYPE
SQLState	5RW07
ODBC State	S0002
Parameter 1	Name of remote message type.

**Probable cause** You have referred to a remote message type that is not defined in this database. CREATE REMOTE TYPE is used to define remote message types.

## Remote statement failed

Item	Value
SQLCode	-288
Constant	SQLE_REMOTE_STATEMENT_FAILED
SQLState	5RW09
ODBC State	S0002

**Probable cause** This SQLSTATE can be signalled within a trigger to prevent DBREMOTE from displaying an error message in the output. This exception will only occur when a trigger or procedure SIGNALS it. This is useful for ignoring replication errors that are permitted by design.

## Request denied -- no active databases

Item	Value
SQLCode	-76
Constant	SQLE_REQUEST_DENIED_NO_DATABASES
SQLState	08W28
ODBC State	08001

**Probable cause** The engine has denied the request as there are currently no loaded databases.

## Request to start/stop database denied

Item	Value
SQLCode	-75
Constant	SQLE_START_STOP_DATABASE_DENIED

Item	Value
SQLState	08W29
ODBC State	08001

**Probable cause**            The engine has denied permission to start/stop a database.

## Result set not allowed from within an atomic compound statement

Item	Value
SQLCode	-222
Constant	SQLE_RESULT_NOT_ALLOWED
SQLState	3BW02
ODBC State	S1000

**Probable cause**            A SELECT statement with no INTO clause or a RESULT CURSOR statement are not allowed within an atomic compound statement.

## Right truncation of string data

Item	Value
SQLCode	-638
Constant	SQLE_STRING_RIGHT_TRUNCATION
SQLState	22001
ODBC State	22001

**Probable cause**            Non-space characters were truncated upon the assignment of string data.

## ROLLBACK TO SAVEPOINT not allowed

Item	Value
SQLCode	-221
Constant	SQLE_ROLLBACK_NOT_ALLOWED
SQLState	3B002

Item	Value
ODBC State	S1000

**Probable cause** A ROLLBACK TO SAVEPOINT within an atomic operation is not allowed to a savepoint established before the atomic operation.

## Row has been updated since last time read

Item	Value
SQLCode	104
Constant	SQLE_ROW_UPDATED_WARNING
SQLState	01W04
ODBC State	(handled by ODBC driver)

**Probable cause** A FETCH has retrieved a row from a cursor declared as a SCROLL cursor, and the row was previously fetched from the same cursor, and one or more columns in the row has been updated since the previous fetch. Note that the column(s) updated may or may not be fetched by the cursor; this warning just indicates that the row from the table has been updated. If the cursor involves more than one table, a row from one or more of the tables has been updated.

## Row has changed since last read -- operation cancelled

Item	Value
SQLCode	-208
Constant	SQLE_ROW_UPDATED_SINCE_READ
SQLState	22W02
ODBC State	(handled by ODBC driver)

**Probable cause** You have done a UPDATE (positioned) or DELETE (positioned) on a cursor declared as a SCROLL cursor, and the row you are changing has been updated since you read it. This prevents the 'lost update' problem.

## Run time SQL error -- %1

Item	Value
SQLCode	-300
Constant	SQLE_ERROR
SQLState	40000
ODBC State	S1000
Parameter 1	Identification of the error.

**Probable cause** This error indicates an internal database error, and should be reported to SQL Anywhere technical support.

## Savepoint '%1' not found

Item	Value
SQLCode	-220
Constant	SQLE_SAVEPOINT_NOTFOUND
SQLState	3B001
ODBC State	S1000
Parameter 1	Name of savepoint.

**Probable cause** You attempted to rollback to a savepoint that does not exist.

## Savepoints require a rollback log

Item	Value
SQLCode	-213
Constant	SQLE_SAVEPOINTS_REQUIRE_UNDO
SQLState	3BW01
ODBC State	S1000

**Probable cause** You cannot use savepoints when the database engine is running in bulk mode without a rollback log.



## SELECT lists in UNION do not match in length

Item	Value
SQLCode	-153
Constant	SQLE_INVALID_UNION
SQLState	53026
ODBC State	37000

### Probable cause

You have specified a UNION but the SELECT statements involved in the union do not have the same number of columns in the select list.

## SELECT returns more than one row

Item	Value
SQLCode	-185
Constant	SQLE_TOO_MANY_RECORDS
SQLState	21000
ODBC State	S1000

### Probable cause

An Embedded SELECT statement that does not use a cursor returns more than one result.

## Server/database engine version mismatch

Item	Value
SQLCode	-232
Constant	SQLE_SERVER_ENGINE_MISMATCH
SQLState	08W20
ODBC State	08001

### Probable cause

Your version of the database server software is not compatible with your version of the database engine.

## **Specified database is invalid**

<b>Item</b>	<b>Value</b>
SQLCode	-84
Constant	SQLE_INVALID_DATABASE
SQLState	08W11
ODBC State	08001

**Probable cause**            The database engine was started but the specified database file is invalid.  
The engine is stopped.

## **Specified database not found**

<b>Item</b>	<b>Value</b>
SQLCode	-83
Constant	SQLE_DATABASE_NOT_FOUND
SQLState	08W10
ODBC State	08001

**Probable cause**            The database engine or multi-user client was started but was unable to find  
the specified database or server name. The database file cannot be opened or  
the specified server cannot be found on the network. The database engine or  
client is stopped.

## **SQL Anywhere HLI internal error**

<b>Item</b>	<b>Value</b>
SQLCode	-406
Constant	SQLE_HLI_INTERNAL
SQLState	WI007
ODBC State	WI007

**Probable cause**            This is a SQL Anywhere internal error and should be reported to Watcom.

## SQL statement error

Item	Value
SQLCode	-132
Constant	SQLE_STATEMENT_ERROR
SQLState	26501
ODBC State	S1000

**Probable cause** The statement identifier (generated by PREPARE) passed to the database for a further operation is invalid.

## Sqllib/dblib version mismatch

Item	Value
SQLCode	-230
Constant	SQLE_PP_DBLIB_MISMATCH
SQLState	08W18
ODBC State	08001

**Probable cause** Your executable has source files with Embedded SQL that were preprocessed with a preprocessor that does not match the database interface library.

## Standard logons are not permitted

Item	Value
SQLCode	-206
Constant	SQLE_INVALID_INTEGRATED_LOGON
SQLState	28W03
ODBC State	28000

**Probable cause** The engine logon type switch (gl) is set to INTEGRATED logon type, and the user has attempted a standard logon.

## Statement cannot be executed

Item	Value
SQLCode	111
Constant	SQLE_CANNOT_EXECUTE_STMT
SQLState	01W08
ODBC State	(handled by ODBC driver)

**Probable cause** You have specified a statement for the PREPARE..WITH EXECUTE statement that cannot be executed. If you specified an output SQLDA, it may contain a DESCRIBE of the prepared statement.

## Statement interrupted by user

Item	Value
SQLCode	-299
Constant	SQLE_INTERRUPTED
SQLState	57014
ODBC State	S1000

**Probable cause** The user has aborted a statement during its execution. The database was able to stop the operation without doing a rollback. If the statement is INSERT, UPDATE, or DELETE, any changes made by the statement will be cancelled. If the statement is a data definition command (for example CREATE TABLE), the command will be cancelled, but the COMMIT that was done as a side effect will not be cancelled.

## Subquery allowed only one select list item

Item	Value
SQLCode	-151
Constant	SQLE_SUBQUERY_SELECT_LIST
SQLState	53023
ODBC State	37000

**Probable cause** You have entered a subquery which has more than one column in the select list. Change the select list to have only one column.

## Subquery cannot return more than one result

Item	Value
SQLCode	-186
Constant	SQLE_SUBQUERY_RESULT_NOT_UNIQUE
SQLState	21W01
ODBC State	37000

**Probable cause** The result of a subquery contains more than one row. If the subquery is in the WHERE clause, you might be able to use IN.

## Subscription to '%1' for '%2' already exists

Item	Value
SQLCode	-282
Constant	SQLE_SUBSCRIPTION_NOT_UNIQUE
SQLState	5RW03
ODBC State	S0002
Parameter 1	Name of the publication.
Parameter 2	Name of the user.

**Probable cause** You have tried to create a subscription that already exists.

## Subscription to '%1' for '%2' not found

Item	Value
SQLCode	-283
Constant	SQLE_SUBSCRIPTION_NOT_FOUND
SQLState	5RW04

Item	Value
ODBC State	S0002
Parameter 1	Name of the publication.
Parameter 2	Name of the user.

**Probable cause** You have tried to drop, start, or synchronize a subscription that does not exist.

### Syntax error near '%1'

Item	Value
SQLCode	-131
Constant	SQLE_SYNTAX_ERROR
SQLState	42W04
ODBC State	37000
Parameter 1	The word or symbol where the syntax error has been detected.

**Probable cause** The database engine cannot understand the command you are trying to execute. If you have used a keyword (such as DATE) for a column name, try enclosing the keyword in quotation marks ("DATE").

### Table '%1' has no primary key

Item	Value
SQLCode	-118
Constant	SQLE_NO_PRIMARY_KEY
SQLState	55008
ODBC State	23000
Parameter 1	Name of the table that does not have a primary key.

**Probable cause** You have attempted to add a foreign key referring to a table that does not have a primary key. You will need to add a primary key to the named table.

## Table '%1' has publications

Item	Value
SQLCode	-281
Constant	SQLE_TABLE_HAS_PUBLICATIONS
SQLState	5RW02
ODBC State	S0002
Parameter 1	Name of the publication that has publications.

**Probable cause** You have attempted to drop a table that has publications defined.

## Table '%1' is in an outer join cycle

Item	Value
SQLCode	-136
Constant	SQLE_OUTER_JOIN_CYCLE
SQLState	52W14
ODBC State	37000
Parameter 1	Name of a table in the cycle.

**Probable cause** You have specified outer joins that create a cycle of tables.

## Table '%1' not found

Item	Value
SQLCode	-141
Constant	SQLE_TABLE_NOT_FOUND
SQLState	42W33
ODBC State	42000
Parameter 1	Name of the table that could not be found.

**Probable cause** You have misspelled the name of a table, or you have connected with a different user ID and forgotten to qualify a table name with a user name. For example, you might have referred to employee instead of "DBA".employee,

### Table '%1' requires a unique correlation name

Item	Value
SQLCode	-137
Constant	SQLE_CORRELATION_NAME_NEEDED
SQLState	52W15
ODBC State	37000
Parameter 1	Name of the table that needs a unique correlation name.

**Probable cause** You have specified a join that joins a table to itself. You need to use unique correlation names in order to have multiple instances of a table.

### Table already has a primary key

Item	Value
SQLCode	-112
Constant	SQLE_EXISTING_PRIMARY_KEY
SQLState	55013
ODBC State	23000

**Probable cause** You have tried to add a primary key on a table that already has a primary key defined. You must delete the current primary key before adding a new one.

### Table cannot have two primary keys

Item	Value
SQLCode	-126
Constant	SQLE_PRIMARY_KEY_TWICE



Item	Value
SQLState	52W05
ODBC State	23000

**Probable cause** You have specified the primary key twice in a CREATE TABLE command.

## Table in use

Item	Value
SQLCode	-214
Constant	SQLE_TABLE_IN_USE
SQLState	42W21
ODBC State	40001

**Probable cause** You have attempted to ALTER or DROP a table that is being used by other active users of the database.

## Table must be empty

Item	Value
SQLCode	-116
Constant	SQLE_TABLE_MUST_BE_EMPTY
SQLState	55W02
ODBC State	S1000

**Probable cause** You have attempted to modify a table, and SQL Anywhere can only perform the change if there are no rows in the table.

## Terminated by user -- transaction rolled back

Item	Value
SQLCode	-302
Constant	SQLE_TERMINATED_BY_USER

<b>Item</b>	<b>Value</b>
SQLState	40W02
ODBC State	S1000

**Probable cause** The user has aborted a command while the database was executing. A ROLLBACK WORK command has been automatically executed. This will happen when the engine is running in bulk mode and the user aborts an INSERT, UPDATE, or DELETE operation.

### **The selected database is currently inactive**

<b>Item</b>	<b>Value</b>
SQLCode	-74
Constant	SQLE_DATABASE_NOT_ACTIVE
SQLState	08W30
ODBC State	08001

**Probable cause** The selected database is in an inactive state. This state occurs during database initialization and shutdown.

### **The supplied buffer was too small to hold all requested query results**

<b>Item</b>	<b>Value</b>
SQLCode	400
Constant	SQLE_HLI_MORE_DATA_AVAILABLE
SQLState	01WH1
ODBC State	(handled by ODBC driver)

**Probable cause** You attempted to get a query result set using the WSQL HLI function wsqquerytomem. The buffer supplied by the calling application was too small to contain the entire query. The buffer will contain as many rows of the result set as possible, and the cursor will be positioned on the next row of the result set.

## There are still active database connections

Item	Value
SQLCode	-109
Constant	SQLE_STILL_ACTIVE_CONNECTIONS
SQLState	08W06
ODBC State	S1000

### Probable cause

An application has requested SQL Anywhere to shutdown the database using the `db_stop()` function when there are still active connections to the database.

## There is already a variable named '%1'

Item	Value
SQLCode	-261
Constant	SQLE_VARIABLE_EXISTS
SQLState	42W15
ODBC State	37000

### Probable cause

You have tried to `CREATE` a variable with the name of another variable that already exists.

## There is more than one way to join '%1' to '%2'

Item	Value
SQLCode	-147
Constant	SQLE_AMBIGUOUS_JOIN
SQLState	52W08
ODBC State	37000
Parameter 1	Name of first table that cannot be joined.
Parameter 2	Name of second table that cannot be joined.

**Probable cause**

There are two or more foreign keys relating the two tables and you are attempting to KEY JOIN the two tables. Either there are two foreign keys from the first table to the second table, or each table has a foreign key to the other table. You must use a correlation name for the primary key table which is the same as the role name of the desired foreign key relationship.

**There is no way to join '%1' to '%2'**

Item	Value
SQLCode	-146
Constant	SQLE_CANNOT_JOIN
SQLState	53W04
ODBC State	37000
Parameter 1	Name of first table that cannot be joined.
Parameter 2	Name of second table that cannot be joined.

**Probable cause**

You have attempted a KEY JOIN between two tables and there is no foreign key on one of the tables that references the primary key of the other table; or you have attempted a NATURAL JOIN between two tables and the tables have no common column names.

**Too many columns in table**

Item	Value
SQLCode	-616
Constant	SQLE_TOO_MANY_COLUMNS_IN_TABLE
SQLState	52W20
ODBC State	S1000

**Probable cause**

A CREATE TABLE or ALTER TABLE statement attempted to add a column to a table, but the resulting number of columns in the table would exceed the limit for the current database page size.

## Too many connections to database

Item	Value
SQLCode	-102
Constant	SQLE_TOO_MANY_CONNECTIONS
SQLState	08W03
ODBC State	08004

### Probable cause

If you are running the multi-user client, you have exceeded the number of computers allowed to connect to the server by your license agreement. Otherwise, the single user DOS engine is limited to 2 connections, and the Windows engine is restricted to 10 connections.

## Too many parameters to this external procedure call

Item	Value
SQLCode	-625
Constant	SQLE_TOO_MANY_PARAMETERS
SQLState	WW010
ODBC State	S1000

### Probable cause

This is a Windows 32-bit specific error. There is a maximum of 256 parameters to an external function call.

## Transact-SQL feature not supported

Item	Value
SQLCode	-611
Constant	SQLE_TSQL_FEATURE_NOT_SUPPORTED
SQLState	0AW02
ODBC State	37000

### Probable cause

An attempt was made to use a feature of Transact-SQL that is not supported.

## Transaction log backup page only partially full

Item	Value
SQLCode	110
Constant	SQLE_BACKUP_PAGE_INCOMPLETE
SQLState	01W10
ODBC State	(handled by ODBC driver)

### Probable cause

A DB\_LOG\_BACKUP\_READ\_WAIT was issued against the transaction log and the page returned was not full. The application should reissue the request for the same page.

## Transaction log was truncated

Item	Value
SQLCode	-244
Constant	SQLE_LOG_TRUNCATED
SQLState	WB005
ODBC State	S1000

### Probable cause

An operation was being performed on the transaction log such as SQL Remote or Replication Agent processing and the transaction log was truncated by an independent backup during that operation.

## Trigger '%1' not found

Item	Value
SQLCode	-268
Constant	SQLE_TRIGGER_NOT_FOUND
SQLState	52W10
ODBC State	S0002
Parameter 1	Name of the trigger that could not be found.

**Probable cause** You have misspelled the name of a trigger, or you have connected with a different userid and forgotten to qualify a trigger name with a user name.

### Trigger definition conflicts with existing triggers

Item	Value
SQLCode	-271
Constant	SQLE_TRIGGER_DEFN_CONFLICT
SQLState	52W11
ODBC State	S0001

**Probable cause** A trigger definition could not be created because it conflicts with an existing trigger definition. A trigger with the same name may already exist.

### Triggers and procedures not supported in runtime engine

Item	Value
SQLCode	-275
Constant	SQLE_PROCEDURES_NOT_IN_DESKTOP
SQLState	0AW04
ODBC State	S1000

**Probable cause** You have attempted to call a stored procedure or have modified a row in a table on which a trigger is defined and you are using the desktop engine. Triggers and stored procedures are not supported in the runtime engine. You must be running the full engine to use these features.

### Unable to delete database file

Item	Value
SQLCode	-243
Constant	SQLE_BACKUP_UNABLE_TO_DELETE_FILE
SQLState	WB004

Item	Value
ODBC State	S1000

**Probable cause** The specified file could not be deleted. The filename should not be the same as any database file that is currently in use.

## Unable to find in index '%1' for table '%2'

Item	Value
SQLCode	-189
Constant	SQLE_NOT_FOUND_IN_INDEX
SQLState	WI005
ODBC State	S1000
Parameter 1	Name of invalid index.
Parameter 2	Name of table containing the invalid index.

**Probable cause** This is a SQL Anywhere internal error and should be reported to SQL Anywhere technical support. You should be able to work around the error by dropping and recreating the index.

## Unable to start database engine

Item	Value
SQLCode	-80
Constant	SQLE_UNABLE_TO_START_ENGINE
SQLState	08W07
ODBC State	08001

**Probable cause** It was not possible to start the database engine or multi-user client. Either there is not enough memory to run the database engine, or the executable cannot be found.



## Unable to start specified database

Item	Value
SQLCode	-82
Constant	SQLE_UNABLE_TO_START_DATABASE
SQLState	08W09
ODBC State	08001

**Probable cause** The database engine or multi-user client was started but was unable to find the specified database or server name. No specific reason is known.

## Unknown backup operation

Item	Value
SQLCode	-240
Constant	SQLE_UNKNOWN_BACKUP_OPERATION
SQLState	WB001
ODBC State	S1000

**Probable cause** An invalid backup command operation was specified in a call to db\_backup.

## Unknown function '%1'

Item	Value
SQLCode	-148
Constant	SQLE_UNKNOWN_FUNC
SQLState	42W05
ODBC State	37000
Parameter 1	Function name that is not a database function.

**Probable cause** You have misspelled the name of a database function (such as MAXIMUM instead of MAX) in a query definition or in a query column name.

## Unterminated C string

Item	Value
SQLCode	-634
Constant	SQLE_UNTERMINATED_C_STR
SQLState	22024
ODBC State	22024

**Probable cause** The least significant character of a C string host variable must contain the null character.

## Update operation attempted on a read-only cursor

Item	Value
SQLCode	-633
Constant	SQLE_READ_ONLY_CURSOR
SQLState	42W30
ODBC State	42030

**Probable cause** An update operation has been attempted on a cursor that was explicitly declared as read-only.

## Update operation attempted on non-updatable query

Item	Value
SQLCode	-192
Constant	SQLE_NON_UPDATEABLE_VIEW
SQLState	42W31
ODBC State	42031

**Probable cause**

You have attempted an insert, update, or delete operation on a query that is implicitly read-only. An updatable query may not contain DISTINCT, GROUP BY, HAVING, or UNION, nor may it contain aggregate functions or involve a join. If the query references a view then the query expression that defines the view must itself be updatable.

**User '%1' already has grant permission**

Item	Value
SQLCode	-120
Constant	SQLE_ALREADY_HAS_GRANT_PERMS
SQLState	42W01
ODBC State	37000
Parameter 1	Name of the userid that already has GRANT permission.

**Probable cause**

The SQL GRANT command is attempting to give a user GRANT OPTION and that user already has GRANT OPTION.

**User '%1' already has membership in group '%2'**

Item	Value
SQLCode	-312
Constant	SQLE_ALREADY_HAS_GROUP_MEMBERSHIP
SQLState	42W34
ODBC State	37000
Parameter 1	Name of the userid that already has membership.
Parameter 2	Name of the group.

**Probable cause**

The SQL GRANT command is attempting to give a membership in a group to user that already has such membership.

## User '%1' has the row in '%2' locked

Item	Value
SQLCode	-210
Constant	SQLE_LOCKED
SQLState	42W18
ODBC State	40001
Parameter 1	Name of another user.
Parameter 2	Table which generates the error.

**Probable cause** You have attempted to read or write a row and it is locked by another user. Note that this error will only be received if the database option **BLOCKING** is set to **OFF**. Otherwise, the requesting transaction will block until the row lock is released.

## User '%1' is already the publisher for this database

Item	Value
SQLCode	-284
Constant	SQLE_ONLY_ONE_PUBLISHER
SQLState	5RW05
ODBC State	S0002
Parameter 1	Name of the publisher.

**Probable cause** You have tried to **GRANT PUBLISH** to a userid, when a publisher already exists.

## User '%1' is not a remote user for this database

Item	Value
SQLCode	-285
Constant	SQLE_NOT_REMOTE_USER
SQLState	5RW06

Item	Value
ODBC State	S0002
Parameter 1	Name of user.

**Probable cause**

You have tried to CREATE a subscription for a user, or PASSTHROUGH for a user that is not a remote user of this database. You must GRANT REMOTE or GRANT CONSOLIDATE.

**User '%1' is not a user group**

Item	Value
SQLCode	-123
Constant	SQLE_NOT_A_GROUP
SQLState	42W03
ODBC State	37000
Parameter 1	Name of user you thought was a group.

**Probable cause**

You have tried to add a member to group, but the group specified has not been granted the GROUP special privilege.

**User message %1 already exists**

Item	Value
SQLCode	-610
Constant	SQLE_MESSAGE_ALREADY_EXISTS
SQLState	52W16
ODBC State	23000

**Probable cause**

The message with this error number already exists in SYSUSERMESSAGES.

## User message %1 not found

Item	Value
SQLCode	-612
Constant	SQLE_MESSAGE_NOT_FOUND
SQLState	52W17
ODBC State	S0002
Parameter 1	Message number.

**Probable cause** The message with this error number does not exist in SYSUSERMESSAGES.

## User-defined exception signalled

Item	Value
SQLCode	-297
Constant	SQLE_USER_DEFINED_EXCEPTION
SQLState	99999
ODBC State	S1000

**Probable cause** A stored procedure or trigger signalled a user-defined exception. This error state is reserved for use within stored procedures or triggers which contain exception handlers, as a way of signalling an exception which can be guaranteed to not have been caused by the database engine.

## User-defined type %1 not found

Item	Value
SQLCode	-613
Constant	SQLE_USER_TYPE_NOT_FOUND
SQLState	52W18
ODBC State	S0002
Parameter 1	Name of the user-defined type.

**Probable cause** The user-defined type with this name does not exist in SYSUSERTYPE.

### Userid '%1' does not exist

Item	Value
SQLCode	-140
Constant	SQLE_UNKNOWN_USERID
SQLState	08004
ODBC State	28000
Parameter 1	Name of the userid that could not be found.

**Probable cause** The specified userid does not exist.

### Using temporary table

Item	Value
SQLCode	102
Constant	SQLE_TEMPORARY_TABLE
SQLState	01W02
ODBC State	(handled by ODBC driver)

**Probable cause** A temporary table has been created in order to satisfy the query. It can only occur on an OPEN statement.

### Value %1 out of range for destination

Item	Value
SQLCode	-158
Constant	SQLE_OVERFLOW_ERROR
SQLState	22003
ODBC State	22003
Parameter 1	The value that caused the overflow.

**Probable cause** A value has been supplied to the database or retrieved from the database that is out of range for the destination column or host variable. For example, the value 10 may have been supplied for a DECIMAL(3,2) field.

## Value for column '%1' in table '%2' has changed

Item	Value
SQLCode	106
Constant	SQLE_COLUMN_VALUE_CHANGED
SQLState	01W06
ODBC State	(handled by ODBC driver)
Parameter 1	Name of the column whose value has changed.
Parameter 2	Name of the table containing the column.

**Probable cause** A replicated UPDATE has found a value in an updated column that does not match the value when the original UPDATE was made.

## Value truncated

Item	Value
SQLCode	101
Constant	SQLE_TRUNCATED
SQLState	01004
ODBC State	01004

**Probable cause** You have tried to insert, update, or select a value in the database which is too large to fit in the destination. This warning is also produced if you do a fetch, and the host variable or SQLDA variable is not large enough to receive the value.



**Variable '%1' not found**

Item	Value
SQLCode	-260
Constant	SQLE_VARIABLE_NOT_FOUND
SQLState	42W14
ODBC State	37000

**Probable cause**

You have tried to DROP or SET the value of a SQL variable that was not created or was previously dropped.

**Warning**

Item	Value
SQLCode	200
Constant	SQLE_WARNING
SQLState	01000
ODBC State	(handled by ODBC driver)

**Probable cause**

A warning has occurred. The warning message will indicate the condition that caused the warning.

**WITH CHECK OPTION violated for view '%1'**

Item	Value
SQLCode	-632
Constant	SQLE_WITH_CHECK_OPTION_VIOLATION
SQLState	44000
ODBC State	44000
Parameter 1	View where check option violated

**Probable cause**

A value in the row(s) being inserted or modified fell outside the range of the view

## Wrong number of parameters to function '%1'

Item	Value
SQLCode	-154
Constant	SQLE_WRONG_PARAMETER_COUNT
SQLState	37505
ODBC State	37000
Parameter 1	Name of the function.

**Probable cause** You have supplied an incorrect number of parameters to a database function.

## Wrong number of values for INSERT

Item	Value
SQLCode	-207
Constant	SQLE_WRONG_NUM_OF_INSERT_COLS
SQLState	53002
ODBC State	37000

**Probable cause** The number of values you are trying to insert does not match the number of columns specified in the INSERT command, or the number of columns in the table if no columns are specified.

## Wrong number of variables in FETCH

Item	Value
SQLCode	-264
Constant	SQLE_WRONG_NUM_OF_FETCH_VARIABLES
SQLState	42W26
ODBC State	37000

**Probable cause** The number of variables specified in the FETCH statement does not match the number of select list items.



## **Internal errors (assertion failed)**

The SQL Anywhere engine has many internal checks that have been designed to detect possible database corruption as soon as possible. If the database engine prints an Assertion Failed message, you should not continue to use it before attempting to determine the cause. You should record the assertion number displayed on the screen and report the error to SQL Anywhere technical support.

The DBVALID utility is useful for determining if your database file is corrupt. You may find it necessary to reconstruct your data from backups and transaction logs (see the chapter "Backup and Recovery").

## CHAPTER 43

# SQL Preprocessor Error Messages

About this chapter

This chapter presents a list of all SQL preprocessor errors and warnings.

Contents

<b>Topic</b>	<b>Page</b>
SQLPP errors	1298
SQLPP warnings	1307

## SQLPP errors

### subscript value %ld too large

Item	Value
Code	2601

Probable cause            You have attempted to index a host variable that is an array with a value too large for the array.

### combined pointer and arrays not supported for hosttypes

Item	Value
Code	2602

Probable cause            You have used an array of pointers as a host variable. This is not legal.

### only one dimensional arrays supported for char type

Item	Value
Code	2603

Probable cause            You have used an array of arrays of character as a host variable. This is not a legal host variable type.

### VARCHAR type must have a length

Item	Value
Code	2604

Probable cause            You have attempted to declare a VARCHAR or BINARY host variable using the DECL\_VARCHAR or DECL\_BINARY macro but have not specified a size for the array.

**arrays of VARCHAR not supported**

Item	Value
Code	2605

**Probable cause** You have attempted to declare a host variable as an array of VARCHAR or BINARY. This is not a legal host variable type.

**VARCHAR host variables cannot be pointers**

Item	Value
Code	2606

**Probable cause** You have attempted to declare a host variable as a pointer to a VARCHAR or BINARY. This is not a legal host variable type.

**initializer not allowed on VARCHAR host variable**

Item	Value
Code	2607

**Probable cause** You can not specify a C variable initializer for a host variable of type VARCHAR or BINARY. You must initialize this variable in regular C executable code.

**FIXCHAR type must have a length**

Item	Value
Code	2608

**Probable cause** You have used the DECL\_FIXCHAR macro to declare a host variable of type FIXCHAR but have not specified a length.

## arrays of FIXCHAR not supported

Item	Value
Code	2609

Probable cause      You have attempted to declare a host variable as an array of FIXCHAR. This is not a legal host variable type.

## arrays of int not supported

Item	Value
Code	2610

Probable cause      You have attempted to declare a host variable as an array of ints. This is not a legal host variable type.

## precision must be specified for decimal type

Item	Value
Code	2611

Probable cause      You must specify the precision when declaring a packed decimal host variable using the DECL\_DECIMAL macro. The scale is optional.

## arrays of decimal not allowed

Item	Value
Code	2612

Probable cause      You have attempted to declare a host variable as an array of DECIMAL. This is not a legal host variable type.



**Unknown hostvar type**

Item	Value
Code	2613

Probable cause You declared a host variable of a type not understood by the SQL preprocessor.

**invalid integer**

Item	Value
Code	2614

Probable cause An integer was required in an embedded SQL statement (for a fetch offset, or a host variable array index, etc.) and the preprocessor was unable to convert what was supplied into an integer.

**'%s' host variable must be a C string type**

Item	Value
Code	2615

Probable cause A C string was required in an embedded SQL statement (for a cursor name, option name etc.) and the value supplied was not a C string.

**'%s' symbol already defined**

Item	Value
Code	2617

Probable cause You defined a host variable twice with different definitions.

## invalid type for sql statement variable

Item	Value
Code	2618

Probable cause      A host variable used as a statement identifier should be of type `_sql_statement_number`. You attempted to use a host variable of some other type as a statement identifier.

## Cannot find include file '%s'

Item	Value
Code	2619

Probable cause      The specified include file was not found. Note that the preprocessor will use the `INCLUDE` environment variable to search for include files.

## host variable '%s' is unknown

Item	Value
Code	2620

Probable cause      You have used a host variable in a statement and that host variable has not been declared in a declare section.

## indicator variable '%s' is unknown

Item	Value
Code	2621

Probable cause      You have used an indicator variable in a statement and that indicator variable has not been declared in a declare section.

**invalid type for indicator variable '%s'**

Item	Value
Code	2622

Probable cause      Indicator variables must be of type short int. You have used a variable of a different type as an indicator variable.

**invalid host variable type on '%s'**

Item	Value
Code	2623

Probable cause      You have used a host variable that is not a string type in a place where the preprocessor was expecting a host variable of a string type.

**host variable '%s' has two different definitions**

Item	Value
Code	2625

Probable cause      The same host variable name was defined with two different types within the same module. Note that host variable names are global to a C module.

**statement '%s' not previously prepared**

Item	Value
Code	2626

Probable cause      An embedded SQL statement name has been used (EXECUTEd) without first being PREPARED.

**cursor '%s' not previously declared**

<b>Item</b>	<b>Value</b>
Code	2627

Probable cause      An embedded SQL cursor name has been used (in a FETCH, OPEN, CLOSE etc.) without first being DECLARED.

**unknown statement '%s'**

<b>Item</b>	<b>Value</b>
Code	2628

Probable cause      You attempted to drop an embedded SQL statement that doesn't exist.

**host variables not allowed for this cursor**

<b>Item</b>	<b>Value</b>
Code	2629

Probable cause      Host variables are not allowed on the declare statement for the specified cursor. If the cursor name is provided through a host variable, then you should use full dynamic SQL and prepare the statement. A prepared statement may have host variables in it.

**host variables specified twice - on declare and open**

<b>Item</b>	<b>Value</b>
Code	2630

Probable cause      You have specified host variables for a cursor on both the declare and the open statements. In the static case, you should specify the host variables on the declare statement. In the dynamic case, specify them on the open.

**must specify a host list or using clause on %s**

Item	Value
Code	2631

**Probable cause** The specified statement requires host variables to be specified either in a host variable list or from an SQLDA.

**no INTO clause on SELECT statement**

Item	Value
Code	2633

**Probable cause** You specified an embedded static SELECT statement but you did not specify an INTO clause for the results.

**incorrect Embedded SQL syntax**

Item	Value
Code	2636

**Probable cause** An embedded SQL specific statement (OPEN, DECLARE, FETCH etc.) has a syntax error.

**missing ending quote of string**

Item	Value
Code	2637

**Probable cause** You have specified a string constant in an embedded SQL statement, but there is no ending quote before the end of line or end of file.

## token too long

Item	Value
Code	2639

### Probable cause

The SQL preprocessor has a maximum token length of 2K. Any token longer than 2K will produce this error. For constant strings in embedded SQL commands (the main place this error shows up) use string concatenation to make a longer string.

## '%s' host variable must be an integer type

Item	Value
Code	2640

### Probable cause

You have used a host variable that is not of integer type in a statement where only an integer type host variable is allowed.

## SQLPP warnings

### Into clause not allowed on declare cursor - ignored

Item	Value
Code	2660

**Probable cause**

You have specified an into clause on a SELECT statement in a declare cursor. Note that the into clause will be ignored.

### unrecognized SQL syntax

Item	Value
Code	2661

**Probable cause**

You have used a SQL statement that will probably cause a syntax error when the statement is sent to the database engine.

### unknown sql function '%s'

Item	Value
Code	2662

**Probable cause**

You have used a SQL function that is unknown to the preprocessor and will probably cause an error when the statement is sent to the database engine.

### wrong number of parms to sql function '%s'

Item	Value
Code	2663

**Probable cause**

You have used a SQL function with the wrong number of parameters. This will likely cause an error when the statement is sent to the database engine.

## static statement names will not work properly if used by 2 threads

Item	Value
Code	2664

### Probable cause

You have used a static statement name and preprocessed with the `-r` reentrancy switch. Static statement names cause static variables to be generated that are filled in by the database. If two threads use the same statement, contention arises over this variable. Use a local host variable as the statement identifier instead of a static name.



## CHAPTER 44

# Differences from Other SQL Dialects

About this chapter      SQL Anywhere conforms to the ANSI SQL89 standard but has many additional features defined in IBM's DB2 and SAA specification, and in ANSI SQL/92.

                                 This chapter describes those features of SQL Anywhere that are not commonly found in other SQL implementations.

Contents	Topic	Page
	SQL Anywhere features	1310

## SQL Anywhere features

The following SQL Anywhere features are not found in many other SQL implementations.

**Type conversions** Full type conversion is implemented. Any data type can be compared with or used in any expression with any other data type.

**Dates** SQL Anywhere has date, time and timestamp types that includes a year, month and day, hour, minutes, seconds and fraction of a second. For insertions or updates to date fields, or comparisons with date fields, a free format date is supported.

In addition, the following operations are allowed on dates:

- ◆ **date + integer** Add the specified number of days to a date.
- ◆ **date - integer** Subtract the specified number of days from a date.
- ◆ **date - date** Compute the number of days between two dates.
- ◆ **date + time** Make a timestamp out of a date and time.

Also, many functions are provided for manipulating dates and times. See "Watcom-SQL Functions" for a description of these.

**Integrity** SQL Anywhere supports both entity and referential integrity. This has been implemented via the following two extensions to the CREATE TABLE and ALTER TABLE commands.

```
PRIMARY KEY ( column-name, ... )
[NOT NULL] FOREIGN KEY [role-name]
    [(column-name, ...)]
    REFERENCES table-name [(column-name, ...)]
    [ CHECK ON COMMIT ]
```

The PRIMARY KEY clause declares the primary key for the relation. SQL Anywhere will then enforce the uniqueness of the primary key, and ensure that no column in the primary key contains the NULL value.

The FOREIGN KEY clause defines a relationship between this table and another table. This relationship is represented by a column (or columns) in this table which must contain values in the primary key of another table. The system will then ensure referential integrity for these columns - whenever these columns are modified or a row is inserted into this table, these columns will be checked to ensure that either one or more is NULL or the values match the corresponding columns for some row in the primary key of the other table. For more information, see "CREATE TABLE statement" in the chapter "Watcom-SQL Statements".

**Joins** SQL Anywhere allows **automatic joins** between tables. In addition to the NATURAL and OUTER join operators supported in other implementations, SQL Anywhere allows KEY joins between tables based on foreign key relationships. This reduces the complexity of the WHERE clause when performing joins.

**Updates** SQL Anywhere allows more than one table to be referenced by the UPDATE command. Views defined on more than one table can also be updated. Many SQL implementations will not allow updates on joined tables.

**Altering tables** The ALTER TABLE command has been extended. In addition to changes for entity and referential integrity, the following types of alterations are allowed:

```
ADD column data-type
MODIFY column data-type
DELETE column
RENAME new-table-name
RENAME old-column TO new-column
```

The MODIFY can be used to change the maximum length of a character column as well as converting from one data type to another. For more information, see "ALTER TABLE statement" in the chapter "Watcom-SQL Statements".

**Subqueries where expressions are allowed**

SQL Anywhere allows subqueries to appear wherever expressions are allowed. Many SQL implementations only allow subqueries on the right side of a comparison operator. For example, the following command is valid in SQL Anywhere but not valid in most other SQL implementations.

```
SELECT      emp_lname,
            emp_birthdate,
            ( SELECT skill
              FROM department
              WHERE emp_id = employee.emp_ID
                AND dept_id = 200 )
FROM employee
```

**Additional functions**

SQL Anywhere supports several functions not in the ANSI SQL definition. See "Watcom-SQL Functions" for a full list of available functions.

**Cursors**

When using Embedded SQL, cursor positions can be moved arbitrarily on the FETCH statement. Cursors can be moved forward or backward relative to the current position or a given number of records from the beginning or end of the cursor.



## CHAPTER 45

# SQL Anywhere Limitations

### About this chapter

This chapter describes the limitations on size and number of objects in SQL Anywhere databases.

### Contents

<b>Topic</b>	<b>Page</b>
Size and number limitations	1314

## Size and number limitations

The following table lists the limitations on size and number of objects in SQL Anywhere database. The memory and disk drive of the microcomputer are more limiting factors in most cases.

Item	Limitation
Database size	2 GB per file, 12 files per database *
Number of tables per database	32,767
Number of tables referenced per transaction	No limit
Table size	2 GB—must be in one file *
Number of columns per table	999 (using 4K pages)
Number of rows per table	Limited by table size
Row size	Limited by table size
Number of rows per database	Limited by file size
Field size	2 GB
Number of indexes	32,767 per table
Maximum index entry size	No limit

- User-created indexes for the table can be stored separately from the table. The 2 GB limit for table size does not apply to Windows NT databases using the NTFS file system. For this platform only, the maximum table size is the largest file size allowed by the operating system.

## CHAPTER 46

# SQL Anywhere Keywords

### About this chapter

This chapter lists SQL keywords reserved by SQL Anywhere. If any of these keywords are to be used as an identifier they should be enclosed in quotation marks.

A number of these keywords are not used in the current implementation of SQL Anywhere but are reserved for future enhancements.

### Contents

The keywords are listed alphabetically.

## Alphabetical list of keywords

If you use a keyword (also called a reserved word) in an identifier you must enclose it in double quotes when referencing it in a SQL statement. Many, but not all, of the words that appear in SQL statements are keywords.

add	all	alter	and
any	as	asc	begin
between	binary	break	by
call	cascade	cast	char
char_convert	character	check	checkpoint
close	comment	commit	connect
constraint	continue	convert	create
cross	current	cursor	date
dba	dbspace	deallocate	dec
decimal	declare	default	delete
desc	distinct	do	double
drop	else	elseif	encrypted
end	endif	escape	exception
exec	execute	exists	fetch
first	float	for	foreign
from	full	goto	grant
group	having	holdlock	identified
if	in	index	inner
inout	insert	instead	int
integer	into	is	isolation
join	key	left	like
lock	long	match	membership
message	mode	modify	named
natural	noholdlock	not	null
numeric	of	off	on
open	option	options	or



order	others	out	outer
passthrough	precision	prepare	primary
print	privileges	proc	procedure
raiserror	readtext	real	reference
references	release	remote	rename
resource	restrict	return	revoke
right	rollback	save	savepoint
schedule	select	set	share
smallint	some	sqlcode	sqlstate
start	stop	subtrans	subtransaction
synchronize	syntax_error	table	temporary
then	time	tinyint	to
tran	trigger	truncate	tsequal
union	unique	unknown	update
user	using	validate	values
varbinary	varchar	variable	varying
view	when	where	while
with	work	writetext	

If you are using Embedded SQL, the database library function **sql\_needs\_quotes** can be used to determine if a string requires quotation marks to make it an identifier because it is a reserved word or contains invalid identifier characters.



## CHAPTER 47

# SQL Anywhere System Procedures and Functions

### About this chapter

This chapter documents the system-supplied catalog stored procedures in SQL Anywhere databases, used to retrieve system information. The chapter also documents system-supplied extended procedures, including procedures for sending e-mail messages on a MAPI e-mail system.

### Contents

<b>Topic</b>	<b>Page</b>
System procedure overview	1320
Catalog stored procedures	1321
System extended stored procedures	1323

## System procedure overview

SQL Anywhere includes the following kinds of system procedures:

- ◆ Catalog stored procedures, for displaying system information in tabular form.
- ◆ Extended stored procedures for MAPI e-mail support and other functions.
- ◆ Transact-SQL system and catalog procedures.

☞ For a list of these system procedures see "SQL Server system and catalog procedures" in the chapter "Using Transact-SQL with SQL Anywhere".

- ◆ System functions that are implemented as stored procedures.

☞ For information see "System functions" in the chapter "Watcom-SQL Functions".

This chapter documents the catalog stored procedures and the extended stored procedures for MAPI e-mail support and other external functions.

## Catalog stored procedures

The following catalog procedures return result sets displaying database engine, database, and connection properties in tabular form.

**sa\_db\_info( [ database-id ] )** Returns a single row containing the Number, Alias, File, ConnCount, PageSize, and LogName for the specified database.

**sa\_conn\_info( [ connection-id ] )** Returns the Number, Name, Userid, DBNumber, LastReqTime, ProcessTime, Port, ReqType, CommLink, NodeAddr, LastIdle, CurrTaskSw, BlockedOn, and UncmtOps properties for each connection. If no connection-id is supplied, information for all current connections to databases on the server is returned.

**sa\_db\_properties( [ database-id ] )** Returns the database id number and the Number, PropNum, PropName, PropDescription, and Value, for each property returned by the sa\_db\_info system procedure.

**sa\_eng\_properties()** Returns the PropNum, PropName, PropDescription, and Value for each available engine property.

☞ For a listing of available engine properties, see "System functions" in the chapter "Watcom-SQL Functions".

**sa\_conn\_properties( [ connection-id ] )** Returns the connection id as Number, and the PropNum, PropName, PropDescription, and Value for each available connection property.

☞ For a listing of available connection properties, see "System functions" in the chapter "Watcom-SQL Functions". These system procedures are owned by the "DBO" user ID. The PUBLIC group has EXECUTE permission on these procedures.

### Examples

- ◆ The following statement returns a single row describing the current database:

```
call sa_db_info
```

Sample values are as follows:

Property	Value
Number	0
Alias	sademo
File	c:\sqlany50\sademo.db

<b>Property</b>	<b>Value</b>
ConnCount	1
PageSize	1024
LogName	c:\sqlany50\sademo.log

- ◆ The following statement returns a set of available engine properties

call sa\_eng\_properties()

<b>PropNum</b>	<b>PropName</b>	<b>...</b>
0	IdleCheck	...
1	IdleWrite	...
2	IdleChkPt	...
...	...	...

## System extended stored procedures

A set of system extended procedures are included into SQL Anywhere databases. These procedures are owned by the DBO user ID that owns the Transact-SQL-compatibility system views.

The following sections describe each of the stored procedures.

### MAPI system extended stored procedures

SQL Anywhere includes a set of three system procedures for sending electronic mail using Microsoft's Messaging API standard (MAPI). These system procedures are implemented as Extended Stored Procedures: each of the procedures calls a function in an external DLL.

In order to use the MAPI stored procedures, a MAPI e-mail system must be accessible from the database server machine.

The MAPI stored procedures are:

- ◆ **xp\_startmail** Starts a mail session in a specified mail account by logging on the MAPI message system
- ◆ **xp\_sendmail** Sends a mail message to specified users
- ◆ **xp\_stopmail** Closes the mail session

The following procedure notifies a set of people that a backup has been completed.

```
CREATE PROCEDURE notify_backup()
BEGIN
    CALL xp_startmail( mail_user='ServerAccount',
                      mail_password='ServerPassword'
                      );
    CALL xp_sendmail( recipient='IS Group',
                      subject='Backup',
                      "message"='Backup completed'
                      );
    CALL xp_stopmail( )
END
```

The MAPI system procedures are discussed in the following sections.

#### xp\_startmail system procedure

**Purpose** To start an e-mail session.

**Syntax** [ [ *variable* = ] CALL ] xp\_startmail (

```
... [ mail_user = mail-login-name ]  
... [, mail_password = mail-password ]  
... )
```

**Usage** Anywhere.

**Authorization** None.

**Description** xp\_startmail is a system stored procedure that starts an e-mail session. It is implemented as a user-defined function.

xp\_startmail returns an integer. The return value is zero if the login is successful, and non-zero otherwise.

The *mail-login-name* and *mail-password* values are strings containing the MAPI login name and password to be used in the mail session.

If you are using Microsoft Exchange, the *mail\_user* argument is an Exchange profile name, and you should not include a password in the procedure call.

**Return codes** The xp\_startmail system procedure uses the following return codes:

Return code	Meaning
0	Success
2	Failure

### xp\_sendmail system procedure

**Purpose** To send an e-mail message.

**Syntax** [ [ *variable* = ] **CALL** ] xp\_sendmail (  
... [ **recipient** = *mail-address* ]  
... [, **cc\_recipient** = *mail-address* ]  
... [, **bcc\_recipient** = *mail-address* ]  
... [, "**message**" = *message-body* ]  
... [, **include\_file** = *file-name* ]  
... )

**Usage** Anywhere.

**Authorization** Must have executed xp\_startmail to start an e-mail session.

**Description** xp\_sendmail is a system stored procedure that sends an e-mail message once a session has been started using :name\_startmail:ename.. It is implemented as a user-defined function.



xp\_sendmail returns an integer. The return value is zero if the message is successfully sent, and non-zero otherwise.

The argument values are strings. The *message* parameter name requires double quotes around it as MESSAGE is a keyword.

**Return codes**

The xp\_sendmail system procedure uses the following return codes:

Return code	Meaning
0	Success
5	Failure (general)
11	Ambiguous recipient
12	Attachment not found
13	Disk full
14	Insufficient memory
15	Invalid session
16	Text too large
17	Too many files
18	Too many recipients
19	Unknown recipient

**Example**

The following call sends a message to the user ID Sales Group containing the file PRICES.DOC as a mail attachment:

```
CALL xp_sendmail(recipient='Sales Group',
                 subject='New Pricing',
                 include_file = 'C:\\DOCS\\PRICES.DOC'
                 )
```

**xp\_stopmail system procedure**

**Purpose** To close an e-mail session.

**Syntax** [ *variable* = ] [ **CALL** ] xp\_stopmail ( )

**Usage** Anywhere.

**Authorization** None.

**Description** xp\_stopmail is a system stored procedure that starts an e-mail session. It is implemented as a user-defined function.

xp\_stopmail returns an integer. The return value is zero if the mail session is successfully closed, and non-zero otherwise.

**Return codes**

The xp\_stopmail system procedure uses the following return codes:

Return code	Meaning
0	Success
3	Failure

## Other system extended stored procedures

The other system extended stored procedures included are:

- ◆ **xp\_cmdshell** Executes a system command.
- ◆ **xp\_sprintf** Takes as input a format string and a set of string arguments, and returns a string.
- ◆ **xp\_scanf** Takes as input a string and a format string, and returns a set of strings.
- ◆ **xp\_msver** Takes as input one of a set of predefined strings, and returns a string containing version information.

The following sections provide more detail on each of these procedures.

### xp\_cmdshell system procedure

<b>Purpose</b>	To carry out an operating system command from a procedure.
<b>Syntax</b>	[ <i>variable</i> = CALL ] xp_cmdshell ( <i>string</i> )
<b>Usage</b>	Anywhere.
<b>Authorization</b>	None.
<b>Description</b>	xp_cmdshell is a system stored procedure that executes a system command and then returns control to the calling environment.
<b>Example</b>	The following statement lists the files in the current directory in the file C:\TEMP.TXT <pre>xp_cmdshell('dir &gt; c:\\temp.txt')</pre>

**xp\_msver system function**

**Purpose** To retrieve version and name information about the database engine or server.

**Syntax** **xp\_msver** ( *string* )  
The string must be one of the following, enclosed in string delimiters.

Argument	Description
ProductName	The name of the product (Sybase SQL Anywhere)
ProductVersion	The version number, followed by the build number. The format is as follows: 5.5.02 (1200)
CompanyName	Returns the following string: Sybase Inc.
FileDescription	Returns the name of the product followed by the name of the operating system.
LegalCopyright	Returns a copyright string for the software
LegalTrademarks	Returns trademark information for the software

**Returns** String containing information appropriate to the argument.

**Usage** Anywhere.

**Authorization** None.

**Description** xp\_msver returns product, company, version, and other information.

**Example** ♦ The following statement requests the version and operating system description:

```
select xp_msver( 'ProductVersion') Version
       xp_msver('FileDescription') Description
```

Sample output is as follows:

Version	Description
5.5.02 (1438)	Sybase SQL Anywhere Windows NT

**xp\_sprintf system procedure**

**Purpose** To build up a string from a format string and a set of input strings.

<b>Syntax</b>	[ <i>variable</i> = <b>CALL</b> ] <b>xp_sprintf</b> ( <i>out-string</i> , ... <i>format-string</i> ... [ <i>input-string</i> , ... ] )
<b>Usage</b>	Anywhere.
<b>Authorization</b>	None.
<b>Description</b>	<p>xp_sprintf is a system stored procedure that builds up a string from a format string and a set of input strings. The format-string can contain up to fifty string placeholders (<i>%s</i>). These placeholders are filled in by the <i>input-string</i> arguments.</p> <p>All arguments must be strings of less than 254 characters.</p>
<b>Example</b>	<p>The following statements put the string <i>Hello World!</i> into the variable <i>mystring</i>.</p> <pre>CREATE VARIABLE mystring CHAR(254) ; xp_sprintf( mystring, 'Hello %s', 'World!' )</pre>

### xp\_scanf system procedure

<b>Purpose</b>	To extract substrings from an input string and a format string.
<b>Syntax</b>	[ <i>variable</i> = <b>CALL</b> ] <b>xp_scanf</b> ( <i>in-string</i> , ... <i>format-string</i> ... [ <i>output-string</i> , ... ] )
<b>Usage</b>	Anywhere.
<b>Authorization</b>	None.
<b>Description</b>	<p>xp_scanf is a system stored procedure that extracts substrings from an input string and a format string. The format-string can contain up to fifty string placeholders (<i>%s</i>). The value of these placeholders in the are filled in by the <i>output-string</i> arguments.</p> <p>All arguments must be strings of less than 254 characters.</p>
<b>Example</b>	<p>The following statements put the string <i>World!</i> into the variable <i>mystring</i>.</p> <pre>CREATE VARIABLE mystring CHAR(254) ; xp_scanf( 'Hello World!', 'Hello %s', mystring )</pre>

## CHAPTER 48

# SQL Anywhere System Tables

### About this chapter

The structure of every SQL Anywhere database is described in a number of **system tables**.

The Entity-Relationship diagram on the next page shows all the system tables and the foreign keys connecting the system tables.

The system tables are owned by the SYS user ID. The contents of these tables can only be changed by the database system. Thus, the UPDATE, DELETE, and INSERT commands cannot be used to modify the contents of these tables. Further, the structure of these tables cannot be changed using the ALTER TABLE and DROP commands.

### Contents

Following a diagram of the system tables, the tables are listed alphabetically.



## Alphabetical list of system tables

This section contains descriptions of each of the system tables. The system tables are described via the CREATE TABLE commands used to create them. They serve as good examples of how tables are created in SQL. Following the CREATE TABLE command, each column is briefly described.

Several of the columns have only two possible values. Usually these values are "Y" and "N" for "yes" and "no" respectively. These columns are designated by "(Y/N)".

### DUMMY system table

```
CREATE TABLE SYS.DUMMY (
    dummy_col      INT NOT NULL
)
```

The DUMMY table is provided as a table that always has exactly one row. This can be useful for extracting information from the database, as in the following example that gets the current user ID and the current date from the database.

```
SELECT USER, today(*) FROM SYS.DUMMY
```

**dummy\_col** This column is not used. It is present because a table cannot be created with no columns.

### SYSARTICLE system table

```
CREATE TABLE SYS.SYSARTICLE (
    publication_id SMALLINT NOT NULL,
    table_id SMALLINT NOT NULL,
    where_expr LONG VARCHAR,
    subscribe_by_expr LONG VARCHAR,
    PRIMARY KEY ( publication_id, table_id ),
    FOREIGN KEY REFERENCES SYS.SYSPUBLICATION,
    FOREIGN KEY REFERENCES SYS.SYSTABLE
)
```

Each row of SYSARTICLE describes an article in a SQL Remote publication.

**publication\_id** The publication of which this article is a part.

**table\_id** Each article consists of columns and rows from a single table. This column contains the table ID for this table.

**where\_expr** For articles that contain a subset of rows defined by a WHERE clause, this column contains the search condition.

**subscribe\_by\_expr** For articles that contain a subset of rows defined by a SUBSCRIBE BY expression, this column contains the expression.

## **SYSARTICLECOL system table**

```
CREATE TABLE SYS.SYSARTICLECOL (  
    publication_id SMALLINT NOT NULL,  
    table_id SMALLINT NOT NULL,  
    column_id SMALLINT NOT NULL,  
    PRIMARY KEY (publication_id, table_id,  
column_id),  
    FOREIGN KEY REFERENCES SYS.SYSARTICLE,  
    FOREIGN KEY REFERENCES SYS.SYSCOLUMN  
)
```

Each row identifies a column in an article, identifying the column, the table it is in, and the publication it is part of.

**publication\_id** A unique identifier for the publication of which the column is a part.

**table\_id** The table to which the column belongs.

**column\_id** The column identifier, from the SYSCOLUMN system table.

## **SYSCOLLATE system table**

```
CREATE TABLE SYS.SYSCOLLATION (  
    collation_id SMALLINT NOT NULL,  
    collation_name CHAR(128) NOT NULL,  
    collation_label CHAR(10) NOT NULL,  
    collation_order BINARY(256) NOT NULL,  
    PRIMARY KEY ( collation_id )  
)
```

This table contains the collation sequences available to SQL Anywhere. There is no way to modify the contents of this table.



**collation\_id** A unique number identifying the collation sequence. The collation sequence with `collation_id` equal 2 is the sequence used in previous versions of SQL Anywhere, and is the default when a database is created with DBINIT.

**collation\_name** The name of the collation sequence.

**collation\_label** A string identifying each of the available collation sequences. The collation sequence to be used is selected when the database is created by specifying the collation label with the `-z` option.

**collation\_order** An array of bytes defining how each of the 256 character codes are treated for comparison purposes. All string comparisons translate each character according to the collation order table before comparing the characters. For the different ASCII code pages, the only difference is how accented characters are sorted. In general, an accented character is sorted as if it were the same as the nonaccented character.

## SYSCOLPERM system table

```
CREATE TABLE SYS.SYSCOLPERM (
    table_id SMALLINT NOT NULL,
    grantee SMALLINT NOT NULL,
    grantor SMALLINT NOT NULL,
    column_id SMALLINT NOT NULL,
    privilege_type SMALLINT NOT NULL,
    is_grantable CHAR( 1 ) NOT NULL,
    PRIMARY KEY ( table_id, grantee,
    grantor, column_id, privilege_type ),
    FOREIGN KEY grantee ( grantee ) REFERENCES
    SYS.SYSUSERPERM ( user_id ),
    FOREIGN KEY grantor ( grantor ) REFERENCES
    SYS.SYSUSERPERM ( user_id ),
    FOREIGN KEY REFERENCES SYS.SYSCOLUMN
)
```

The GRANT command can give UPDATE permission to individual columns in a table. Each column with UPDATE permission is recorded in one row of SYSCOLPERM.

**table\_id** The table number for the table containing the column.

**grantee** The user number of the user ID given UPDATE permission on the column. If the grantee is the user number for the special PUBLIC user ID, the UPDATE permission is given to all user IDs.

**grantor** The user number of the user ID granting the permission.

**column\_id** This column number, together with the table\_id, identifies the column for which UPDATE permission has been granted.

**privilege\_type** The number in this column indicates the kind of column permission (REFERENCES, SELECT or UPDATE).

**is\_grantable** (Y/N). Indicates if the permission on the column was granted by the grantor to the grantee WITH GRANT OPTION.

## **SYSCOLUMN system table**

```
CREATE TABLE SYS.SYSCOLUMN (  
    table_id SMALLINT NOT NULL,  
    column_id SMALLINT NOT NULL,  
    pkey CHAR(1) NOT NULL,  
    domain_id SMALLINT NOT NULL,  
    nulls CHAR(1) NOT NULL,  
    width SMALLINT NOT NULL,  
    scale SMALLINT NOT NULL,  
    estimate INT NOT NULL,  
    column_name CHAR(128) NOT NULL,  
    remarks LONG VARCHAR,  
    "default" LONG VARCHAR,  
    "check" LONG VARCHAR,  
    user_type SMALLINT,  
    format_str CHAR(128),  
    PRIMARY KEY ( table_id, column_id ),  
    FOREIGN KEY REFERENCES SYS.SYSTABLE,  
    FOREIGN KEY REFERENCES SYS.SYSDOMAIN  
)
```

Each column in every table or view is described by one row in SYSCOLUMN.

**table\_id** The table number uniquely identifies the table or view to which this column belongs.

**column\_id** Each table starts numbering columns at 1. The order of column numbers determines the order that columns are displayed in the command select \* from table.

**pkey (Y/N)** Indicate whether this column is part of the primary key for the table.

**domain\_id** Identify the data type for the column by the data type number listed in the SYSDOMAIN table.

**nulls (Y/N)** Indicate whether the NULL value is allowed in this column.

**width** This column contains the length of string columns, the precision of numeric columns, and the number of bytes of storage for all other data types.

**scale** The number of digits after the decimal point for numeric data type columns, and zero for all other data types.

**estimate** A self-tuning parameter for the optimizer. SQL Anywhere will learn from previous queries by adjusting guesses that are made by the optimizer.

**column\_name** The name of the column.

**remarks** A comment string.

**default** The default value for the column. This value is only used when an INSERT statement does not specify a value for the column.

**check.** Any CHECK condition defined on the column.

**user\_type** If the column is defined on a user-defined data type, the data type is held here.

**format\_str** Currently unused.

## SYSDOMAIN system table

```
CREATE TABLE SYS.SYSDOMAIN (
    domain_id      SMALLINT NOT NULL,
    domain_name    CHAR(128) NOT NULL,
    type_id        SMALLINT,
    precision      SMALLINT,
    PRIMARY KEY ( domain_id )
)
```

Each of the predefined data types (sometimes called **domains**) in SQL Anywhere is assigned a unique number. The SYSDOMAIN table is provided for informational purposes to show the association between these numbers and the appropriate data type. This table is never changed by SQL Anywhere.

**domain\_id** The unique number assigned to each data type. These numbers cannot be changed.

**domain\_name** A string containing the data type normally found in the CREATE TABLE command, such as char or integer.

**type\_id** The ODBC data type. This corresponds to "data\_type" in the Transact-SQL-compatibility DBO.SYSTYPES table.

**precision** The number of significant digits that can be stored using this data type. The column value is NULL for non-numeric data types.

## **SYSFILE system table**

```
CREATE TABLE SYS.SYSFILE (
    file_id SMALLINT NOT NULL,
    file_name CHAR(80) NOT NULL,
    dbspace_name CHAR(128) NOT NULL,
    PRIMARY KEY ( file_id )
)
```

Every database consists of one or more operating system files. Each file is recorded in SYSFILE.

**file\_id** Each file in a database is assigned a unique number. This file identifier is the primary key for SYSFILE. All system tables are stored in file\_id 0.

**file\_name** The database name is stored when a database is created. This name is for informational purposes only.

**dbspace\_name** Every file has a dbspace name that is unique. It is used in the CREATE TABLE command.

## **SYSFKCOL system table**

```
CREATE TABLE SYS.SYSFKCOL (
    foreign_table_id SMALLINT NOT NULL,
    foreign_key_id SMALLINT NOT NULL,
    foreign_column_id SMALLINT NOT NULL,
    primary_column_id SMALLINT NOT NULL,
    PRIMARY KEY ( foreign_table_id,
    foreign_key_id, foreign_column_id ),
    FOREIGN KEY REFERENCES SYS.SYSFOREIGNKEY,
    FOREIGN KEY ( foreign_table_id,
    foreign_column_id ) REFERENCES
    SYS.SYSCOLUMN ( table_id, column_id )
)
```

Each row of SYSFKCOL describes the association between a foreign column in the foreign table of a relationship and the primary column in the primary table.

**foreign\_table\_id** The table number of the foreign table.

**foreign\_key\_id** The key number of the FOREIGN KEY for the foreign table. Together, foreign\_table\_id and foreign\_key\_id uniquely identify one row in SYSFORIGNKEY, and the table number for the primary table can be obtained from that row.

**foreign\_column\_id** This column number, together with the foreign\_table\_id, identify the foreign column description in SYSCOLUMN.

**primary\_column\_id** This column number, together with the primary\_table\_id obtained from SYSFORIGNKEY, identify the primary column description in SYSCOLUMN.

## SYSFORIGNKEY system table

```
CREATE TABLE SYS.SYSFORIGNKEY (
    foreign_table_id SMALLINT NOT NULL,
    foreign_key_id SMALLINT NOT NULL,
    primary_table_id SMALLINT NOT NULL,
    root INT NOT NULL,
    check_on_commit CHAR(1) NOT NULL,
    nulls CHAR(1) NOT NULL,
    role CHAR(128) NOT NULL,
    remarks LONG VARCHAR,
    PRIMARY KEY ( foreign_table_id, foreign_key_id ),
    UNIQUE ( role, foreign_table_id ),
    FOREIGN KEY foreign_table ( foreign_table_id )
    REFERENCES SYS.SYSTABLE ( table_id ),
    FOREIGN KEY primary_table ( primary_table_id )
    REFERENCES SYS.SYSTABLE ( table_id )
)
```

A foreign key is a **relationship** between two tables—the **foreign table** and the **primary table**. Every foreign key is defined by one row in SYSFORIGNKEY and one or more rows in SYSFKCOL. SYSFORIGNKEY contains general information about the foreign key while SYSFKCOL identifies the columns in the foreign key and associates each column in the foreign key with a column in the primary key of the primary table.

**foreign\_table\_id** The table number of the foreign table.

**foreign\_key\_id** Each foreign key has a **foreign key number** that is unique with respect to:

- ◆ The key number of all other foreign keys for the foreign table
- ◆ The key number of all foreign keys for the primary table
- ◆ The index number of all indexes for the foreign table

**primary\_table\_id** The table number of the primary table.

**root** Foreign keys are stored in the database as B-trees. The root identifies the location of the root of the B-tree in the database file.

**nulls (Y/N)** Indicate whether the columns in the foreign key are allowed to contain the NULL value. Note that this setting is independent of the nulls setting in the columns contained in the foreign key.

**check\_on\_commit (Y/N)** Indicate whether INSERT and UPDATE commands should wait until the next COMMIT command to check if foreign keys are valid. A foreign key is valid if, for each row in the foreign table, the values in the columns of the foreign key either contain the NULL value or match the primary key values in some row of the primary table.

**role** The name of the relationship between the foreign table and the primary table. Unless otherwise specified, the role name will be the same as the name of the primary table. The foreign table cannot have two foreign keys with the same role name.

**remarks** A comment string.

## **SYSGROUP system table**

```
CREATE TABLE SYS.SYSGROUP (  
    group_id SMALLINT NOT NULL,  
    group_member SMALLINT NOT NULL,  
    PRIMARY KEY ( group_id, group_member ),  
    FOREIGN KEY group_id ( group_id ) REFERENCES  
    SYS.SYSUSERPERM ( user_id ),  
    FOREIGN KEY group_member ( group_member )  
    REFERENCES SYS.SYSUSERPERM ( user_id )  
)
```

There is one row in SYSGROUP for every member of every group. This table describes a many-to-many relationship between groups and members. A group may have many members and a user may be a member of many groups.

**group\_id** The user number of group.

**group\_member** The user number of a member.

## **SYSINDEX system table**

```
CREATE TABLE SYS.SYSINDEX (  

```

```

    table_id SMALLINT NOT NULL,
    index_id SMALLINT NOT NULL,
    root INT NOT NULL,
    file_id SMALLINT NOT NULL,
    "unique" CHAR(1) NOT NULL,
    creator SMALLINT NOT NULL,
    index_name CHAR(128) NOT NULL,
    remarks LONG VARCHAR,
    PRIMARY KEY ( table_id, index_id ),
    UNIQUE ( index_name, creator ),
    FOREIGN KEY REFERENCES SYS.SYSTABLE,
    FOREIGN KEY REFERENCES SYS.SYSFILE,
    FOREIGN KEY ( creator ) REFERENCES
    SYS.SYSUSERPERM ( user_id )
)

```

Each index in the database is described by one row in `SYSINDEX`. Each column in the index is described by one row in `SYSIXCOL`.

**table\_id** The table number uniquely identifies the table to which this index applies.

**index\_id** Each index for one particular table is assigned a unique index number.

**root** Indexes are stored in the database as B-trees. The root identifies the location of the root of the B-tree in the database file.

**file\_id** The index is completely contained in the file with this `file_id` (see `SYSFILE`). In the current implementation of SQL Anywhere, this file is always the same as the file containing the table.

**unique** Indicate whether the index is a unique index ("Y"), a non-unique index ("N"), or a unique constraint ("U"). A unique index prevents two rows in the indexed table from having the same values in the index columns.

**creator** The user number of the creator of the index. In the current implementation of SQL Anywhere, this user is always the same as the creator of the table identified by `table_id`.

**index\_name** The name of the index. A user ID cannot have two indexes with the same name.

**remarks** A comment string.

## SYSINFO system table

```

CREATE TABLE SYS.SYSINFO (
    page_size SMALLINT NOT NULL,

```

```
encryption CHAR(1) NOT NULL,  
blank_padding CHAR(1) NOT NULL,  
case_sensitivity CHAR(1) NOT NULL,  
default_collation CHAR(10) NOT NULL,  
database_version SMALLINT NOT NULL  
)
```

This table indicates the database characteristics as defined when the database was created using DBINIT. It always contains only one row.

**page\_size** The page size specified to DBINIT. The default value is 1024.

**encryption** The value "Y" or "N" depending on whether the -e switch was used with DBINIT.

**blank\_padding** The value "Y" or "N" depending on whether the database was created to use blank padding for string comparisons in the database (-b switch was used with DBINIT).

**case\_sensitivity** The value "Y" or "N" depending on whether the -c switch was used with DBINIT. Case sensitivity affects value comparisons, but not table and column name comparisons. For example, if case sensitivity is enabled, the system catalog names such as SYSCATALOG must be specified in uppercase since that is how the name was spelled when it was created.

**default\_collation** A string corresponding to the collation\_label in SYSCOLLATE corresponding to the collation sequence specified with DBINIT. The default value corresponds to the multilingual collation sequence (code page 850), which was the default prior to Watcom SQL 3.2. The collation sequence is used for all string comparisons, including searches for character strings as well as column and table name comparison.

**database\_version** A small integer value indicating the database format. As newer versions of SQL Anywhere become available, new features may require that the format of the database file change. The version number allows SQL Anywhere software to determine if this database was created with a newer version of the software and thus cannot be understood by the software in use.

## **SYSIXCOL system table**

```
CREATE TABLE SYS.SYSIXCOL (  
  table_id SMALLINT NOT NULL,  
  index_id SMALLINT NOT NULL,  
  sequence SMALLINT NOT NULL,  
  column_id SMALLINT NOT NULL,
```



```

"order" CHAR(1) NOT NULL,
PRIMARY KEY ( table_id, index_id, sequence )
FOREIGN KEY REFERENCES SYS.SYSINDEX,
FOREIGN KEY REFERENCES SYS.SYSCOLUMN
)

```

Every index has one row in SYSIXCOL for each column in the index.

**table\_id** Identifies the table to which the index applies.

**index\_id** Identifies in which index this column is used. Together, table\_id and index\_id identify one index described in SYSINDEX.

**sequence** Each column in an index is assigned a unique number starting at 0. The order of these numbers determines the relative significance of the columns in the index. The most important column has sequence number 0.

**column\_id** The column number identifies which column is indexed. Together, table\_id and column\_id identify one column in SYSCOLUMN.

**order (A/D)** Indicate whether this column in the index is kept in ascending or descending order.

## SYSOPTION system table

```

CREATE TABLE SYS.SYSOPTION (
  user_id SMALLINT NOT NULL,
  "option" CHAR(128) NOT NULL,
  "setting" CHAR(80) NOT NULL,
  PRIMARY KEY ( user_id, "option" ),
  FOREIGN KEY REFERENCES SYS.SYSUSERPERM
)

```

Options settings are stored in the SYSOPTION table by the SET command. Each user can have their own setting for each option. In addition, settings for the PUBLIC user ID define the default settings to be used for user IDs that do not have their own setting.

**user\_id** The user number to whom this option setting applies.

**option** The name of the option.

**setting** The current setting for the named option.

## SYSPROCEDURE system table

```

CREATE TABLE SYS.SYSPROCEDURE (
  proc_id SMALLINT NOT NULL,

```

```
creator SMALLINT NOT NULL,  
proc_name CHAR(128) NOT NULL,  
proc_defn LONG VARCHAR,  
remarks LONG VARCHAR,  
replicate CHAR(1) NOT NULL,  
PRIMARY KEY ( proc_id ),  
UNIQUE ( proc_name, creator ),  
FOREIGN KEY ( creator ) REFERENCES  
SYS.SYSUSERPERM ( user_id )  
)
```

Each procedure in the database is described by one row in SYSPROCEDURE.

**proc\_id** Each procedure is assigned a unique number (the **procedure number**) that is the primary key for SYSPROCEDURE.

**creator** This user number identifies the owner of the procedure. The name of the user can be found by looking in SYSUSERPERM.

**proc\_name** The name of the procedure. One creator cannot have two procedures with the same name.

**proc\_defn** The command used to create the procedure.

**remarks** A comment string.

**replicate** Holds a Y if the procedure is a primary data source in a Replication Server installation, or an N if not.

## SYSPROCPARM system table

```
CREATE TABLE SYS.SYSPROCPARM (  
proc_id SMALLINT NOT NULL,  
parm_id SMALLINT NOT NULL,  
parm_type SMALLINT NOT NULL,  
parm_mode_in CHAR(1) NOT NULL,  
parm_mode_out CHAR(1) NOT NULL,  
domain_id SMALLINT NOT NULL,  
width SMALLINT NOT NULL,  
scale SMALLINT NOT NULL,  
parm_name CHAR(128) NOT NULL,  
remarks LONG VARCHAR,  
"default" LONG VARCHAR,  
PRIMARY KEY ( proc_id, parm_id ),  
FOREIGN KEY REFERENCES SYS.SYSPROCEDURE,  
FOREIGN KEY REFERENCES SYS.SYSDOMAIN  
)
```

Each parameter to a procedure in the database is described by one row in SYSPROCEDURE.

**proc\_id** The procedure number uniquely identifies the procedure to which this parameter belongs.

**parm\_id** Each procedure starts numbering parameters at 1. The order of parameter numbers corresponds to the order in which they were defined.

**parm\_type** The type of parameter will be one of the following:

- ◆ **0-variable** normal parameter
- ◆ **1-result** result variable - used with procedure that return result sets
- ◆ **2-SQLSTATE** SQLSTATE error value
- ◆ **3-SQLCODE** SQLCODE error value

**parm\_mode\_in (Y/N)** Indicate whether this parameter supplies a value to the procedure (IN or INOUT parameters).

**parm\_mode\_out (Y/N)** Indicate whether this parameter returns a value from the procedure (OUT or INOUT parameters).

**domain\_id** Identify the data type for the parameter by the data type number listed in the SYSDOMAIN table.

**width** This column contains the length of string parameters, the precision of numeric parameters, and the number of bytes of storage for all other data types.

**scale** The number of digits after the decimal point for numeric data type parameters, and zero for all other data types.

**parm\_name** The name of the parameter.

**remarks** A comment string.

**default** The default value for the parameter, held as a string.

## SYSPROCPERM system table

```
CREATE TABLE SYS.SYSPROCPERM (
    proc_id SMALLINT NOT NULL,
    grantee SMALLINT NOT NULL,
    PRIMARY KEY ( proc_id, grantee )
    FOREIGN KEY ( grantee ) REFERENCES
    SYS.SYSUSERPERM ( user_id ),
```

```
        FOREIGN KEY REFERENCES SYS.SYSPROCEDURE
    )
```

Only users who have been granted permission can call a procedure. Each row of the SYSPROCPERM table corresponds to one user granted permission to call one procedure.

**proc\_id** The procedure number uniquely identifies the procedure for which permission has been granted.

**grantee** The user number of the user ID receiving the permission.

## **SYSPUBLICATION system table**

```
CREATE TABLE SYS.SYSPUBLICATION (
    publication_id SMALLINT NOT NULL,
    creator SMALLINT NOT NULL,
    publication_name CHAR(128) NOT NULL,
    remarks LONG VARCHAR,
    PRIMARY KEY ( publication_id ),
    FOREIGN KEY ( creator )
    REFERENCES SYS.SYSUSERPERM (user_id )
)
```

Each row describes a SQL Remote publication.

**publication\_id** A unique identifying number for the publication.

**creator** The owner of the publication.

**publication\_name** The name of the publication, which must be a valid identifier.

**remarks** Descriptive comments.

## **SYSREMOTEUSER system table**

```
CREATE TABLE SYS.SYSREMOTEUSER (
    user_id          SMALLINT NOT NULL,
    consolidate     CHAR(1) NOT NULL,
    type_id         SMALLINT NOT NULL,
    address         LONG VARCHAR NOT NULL,
    frequency       CHAR(!) NOT NULL,
    send_time       TIME,
    log_send        NUMERIC(20,0) NOT NULL,
    time_sent       TIMESTAMP,
    log_sent        NUMERIC(20,0) NOT NULL,
    confirm_sent    NUMERIC(20,0) NOT NULL,
    send_count      INTEGER NOT NULL,
```

```

        resend_count          INTEGER NOT NULL,
        time_received         TIMESTAMP,
        log_received          NUMERIC(20,0) NOT NULL,
        confirm_received      NUMERIC(20,0),
        receive_count         INTEGER NOT NULL,
        rereceive_count       INTEGER NOT NULL,
        PRIMARY KEY (user_id),
        FOREIGN KEY REFERENCES SYS.SYSUSERPERM
    )

```

Each row describes a userid with REMOTE permissions (a subscriber), together with the status of SQL Remote messages sent to and from that user.

**user\_id** The user ID of the user with REMOTE permissions.

**consolidate** The column contains either an R to indicate a user granted REMOTE permissions, or a C to indicate a user granted CONSOLIDATE permissions.

**type\_id** Identifies which of the of the message systems supported by SQL Remote is to be used to send messages to this user.

**address** The address to which SQL Remote messages are to be sent. The address must be appropriate for the address\_type.

**frequency** How frequently SQL Remote messages are to be sent.

**send\_time** The next time messages are to be sent to this user.

**log\_send** Messages are sent only to subscribers for whom log\_send is greater than log\_sent.

**time\_sent** The time the most recent message was sent to this subscriber.

**log\_sent** The log offset for the most recently sent operation.

**confirm\_sent** The log offset for the most recently confirmed operation from this subscriber.

**send\_count** How many SQL Remote messages have been sent.

**resend\_count** Counter to ensure messages are applied only once at the subscriber database.

**time\_received** The time the most recent message was received from this subscriber.

**log\_received** The log offset in the subscriber's database for the operation most recently received at the current database.

**confirm\_received** The log offset in the subscriber's database for the most recent operation for which a confirmation message has been sent.

**receive\_count** How many messages have been received.

**rereceive\_count** Counter to ensure messages are applied only once at the current database.

## **SYSSUBSCRIPTION system table**

```
CREATE TABLE SYS.SYSSUBSCRIPTION (  
    publication_id SMALLINT NOT NULL,  
    user_id SMALLINT NOT NULL,  
    subscribe_by CHAR(128) NOT NULL,  
    created NUMERIC(20,0) NOT NULL,  
    started NUMERIC(20,0),  
    PRIMARY KEY (publication_id, user_id,  
    subscribe_by),  
    FOREIGN KEY REFERENCES SYS.SYSPUBLICATION,  
    FOREIGN KEY REFERENCES SYS.SYSREMOTEUSER  
);
```

Each row describes a subscription from one user ID (which must have REMOTE permissions) to one publication.

**publication\_id** The identifier for the publication to which the user ID is subscribed.

**user\_id** The user ID that is subscribed to the publication.

**subscribe\_by** For publications with a SUBSCRIBE BY expression, this column holds the matching value for this subscription.

**created** The offset in the transaction log at which the subscription was created.

**started** The offset in the transaction log at which the subscription was started.

## **SYSTABLE system table**

```
CREATE TABLE SYS.SYSTABLE (  
    table_id SMALLINT NOT NULL,  
    file_id SMALLINT NOT NULL,  
    count INTEGER NOT NULL,  
    first_page INT NOT NULL,  
    last_page INT NOT NULL,  
    primary_root INT NOT NULL,
```

```
creator SMALLINT NOT NULL,  
table_name CHAR(128) NOT NULL,  
table_type CHAR(10) NOT NULL,  
view_def LONG VARCHAR,  
remarks LONG VARCHAR,  
replicate CHAR(1) NOT NULL,  
PRIMARY KEY ( table_id ),  
UNIQUE ( table_name, creator ),  
FOREIGN KEY ( creator ) REFERENCES  
SYS.SYSUSERPERM ( user_id ),  
FOREIGN KEY REFERENCES SYS.SYSFILE  
)
```

Each row of SYSTABLE describes one table or view in the database.

**table\_id** Each table or view is assigned a unique number (the table number) that is the primary key for SYSTABLE.

**file\_id** The file number indicates which database file contains the table. The file\_id is a FOREIGN KEY for SYSFILE.

**count** The number of rows in the table is updated during each successful CHECKPOINT. This number is used by SQL Anywhere when optimizing database access. The count is always 0 for a view.

**first\_page** Each SQL Anywhere database is divided into a number of fixed size pages. This value identifies the first page containing information for this table, and is used internally to find the start of this table. The first\_page is always 0 for a view.

**last\_page** The last page containing information for this table. The last\_page is always 0 for a view.

**primary\_root** Primary keys are stored in the database as B-trees. The primary\_root locates the root of the B-tree for the primary key for the table. It will be 0 for a view and for a table with no primary key.

**creator** This user number identifies the owner of the table or view. The name of the user can be found by looking in SYSUSERPERM.

**table\_name** The name of the table or view. One creator cannot have two tables or views with the same name.

**table\_type** This column will be "BASE" for base tables and "VIEW" for views. It will be "GBL TEMP" for global temporary tables and "LCL TEMP" for local temporary tables.

**view\_def** For a view, this column contains the create view command used to create the view. For a table, this column will contain any CHECK constraints for the table.

**remarks** A comment string.

**replicate** Holds a Y if the table is a primary data source in a Replication Server installation, or an N if not.

## SYSTABLEPERM system table

```
CREATE TABLE SYS.SYSTABLEPERM (
  stable_id SMALLINT NOT NULL,
  grantee SMALLINT NOT NULL,
  grantor SMALLINT NOT NULL,
  ttable_id SMALLINT NOT NULL,
  selectauth CHAR(1) NOT NULL,
  insertauth CHAR(1) NOT NULL,
  deleteauth CHAR(1) NOT NULL,
  updateauth CHAR(1) NOT NULL,
  updatecols CHAR(1) NOT NULL,
  alterauth CHAR(1) NOT NULL,
  referenceauth CHAR(1) NOT NULL,
  PRIMARY KEY ( stable_id, grantee, grantor ),
  FOREIGN KEY ( stable_id )
  REFERENCES SYS.SYSTABLE ( table_id ),
  FOREIGN KEY future ( ttable_id )
  REFERENCES SYS.SYSTABLE ( table_id ),
  FOREIGN KEY grantee ( grantee ) REFERENCES
  SYS.SYSUSERPERM ( user_id ),
  FOREIGN KEY grantor ( grantor )
  REFERENCES SYS.SYSUSERPERM ( user_id )
)
```

Permissions given by the GRANT command are stored in SYSTABLEPERM. Each row in this table corresponds to one table, one user ID granting the permission (**grantor**) and one user ID granted the permission (**grantee**).

There are several types of permission that can be granted. Each permission can have one of the following three values.

- ◆ **N** No, the grantee has not been granted this permission by the grantor.
- ◆ **Y** Yes, the grantee has been given this permission by the grantor.
- ◆ **G** The grantee has been given this permission. In addition, the grantee can grant the same permission to another user



**Permissions**

The grantee might have been given permission for the same table by another grantor. If so, this information would be recorded in a different row of SYSTABLEPERM.

**stable\_id** The table number of the table or view to which the permissions apply.

**grantor** The user number of the user ID granting the permission.

**grantee** The user number of the user ID receiving the permission.

**ttable\_id** In the current version of SQL Anywhere, this table number is always the same as stable\_id.

**selectauth (Y/N/G)** Indicate whether SELECT permission has been granted.

**insertauth (Y/N/G)** Indicate whether INSERT permission has been granted.

**deleteauth (Y/N/G)** Indicate whether DELETE permission has been granted.

**updateauth (Y/N/G)** Indicate whether UPDATE permission has been granted for all columns in the table. (Only UPDATE permission can be given on individual columns. All other permissions are for all columns in a table.)

**updatecols (Y/N)** Indicates whether UPDATE permission has only been granted for some of the columns in the table. If updatecols has the value Y, there will be one or more rows in SYSCOLPERM granting update permission for the columns in this table.

**alterauth (Y/N/G)** Indicate whether ALTER permission has been granted.

**referenceauth (Y/N/G)** Indicate whether REFERENCE permission has been granted.

**SYSTRIGGER system table**

```
CREATE TABLE SYS.SYSTRIGGER (
  trigger_id SMALLINT NOT NULL,
  table_id SMALLINT NOT NULL,
  event CHAR(1) NOT NULL,
  trigger_time CHAR(1) NOT NULL,
```

```
trigger_order SMALLINT,  
foreign_table_id SMALLINT,  
foreign_key_id SMALLINT,  
referential_action CHAR(1),  
trigger_name CHAR(128),  
trigger_defn LONG VARCHAR NOT NULL,  
remarks LONG VARCHAR,  
PRIMARY KEY ( trigger_id ),  
UNIQUE ( trigger_name ),  
UNIQUE ( table_id, event,  
trigger_time, trigger_order ),  
UNIQUE ( table_id, foreign_table_id,  
foreign_key_id, event ),  
FOREIGN KEY REFERENCES SYS.SYSTABLE,  
FOREIGN KEY REFERENCES SYS.SYSFOREIGNKEY  
)
```

Each trigger in the database is described by one row in SYSTRIGGER. The table also contains triggers automatically created by the database for foreign key definitions which have a referential triggered action (such as ON DELETE CASCADE).

**trigger\_id** Each trigger is assigned a unique number (the **trigger number**) that is the primary key for SYSTRIGGER.

**table\_id** The table number uniquely identifies the table to which this trigger belongs.

**event** The event or events that cause the trigger to fire. This single character value corresponds to the trigger event that was specified when the trigger was created.

**trigger\_time** The time at which the trigger will fire. This single character value corresponds to the trigger time that was specified when the trigger was created.

- ◆ **A** AFTER
- ◆ **B** BEFORE

**trigger\_order** The order in which the trigger will fire. This determines the order that triggers are fired when there are triggers of the same type (insert, update, or delete) that fire at the same time (before or after).

**foreign\_table\_id** The foreign table number identifies the table containing a foreign key definition which has a referential triggered action (such as ON DELETE CASCADE).

**foreign\_key\_id** The foreign key number identifies the foreign key for the table referenced by foreign\_table\_id.

**referential\_action** The action defined by a foreign key. This single character value corresponds to the action that was specified when the foreign key was created.

- ◆ **C** CASCADE
- ◆ **D** SET DEFAULT
- ◆ **N** SET NULL
- ◆ **R** RESTRICT

**trigger\_name** The name of the trigger. One table cannot have two triggers with the same name.

**trigger\_defn** The command used to create the trigger.

**remarks** A comment string.

## SYSUSERMESSAGES system table

```
CREATE TABLE SYS.SYSUSERMESSAGES (
    error INT NOT NULL,
    uid SMALLINT NOT NULL,
    description VARCHAR(255) NOT NULL,
    langid SMALLINT NOT NULL,
    UNIQUE ( error, langid )
)
```

Each row holds a user-defined message for an error condition.

**error** A unique identifying number for the error condition.

**uid** The user ID defining the message.

**description** The message corresponding to the error condition.

**langid** Reserved.

## SYSUSERPERM system table

```
CREATE TABLE SYS.SYSUSERPERM (
    user_id SMALLINT NOT NULL,
    user_name CHAR(128) NOT NULL UNIQUE,
    password CHAR(128),
    resourceauth CHAR(1) NOT NULL,
    dbaauth CHAR(1) NOT NULL,
    scheduleauth CHAR(1) NOT NULL,
    publishauth CHAR(1) NOT NULL,
```

```
remotedbaauth    CHAR(1) NOT NULL,  
user_group    CHAR(1) NOT NULL,  
remarks    LONG VARCHAR,  
PRIMARY KEY ( user_id )  
)
```

**DBA permissions required**

SYSUSERPERM contains passwords, and so DBA permissions are required to SELECT from the table.

Each row of SYSUSERPERM describes one user ID.

**user\_id** Each new user ID is assigned a unique number (the **user number**) that is the primary key for SYSUSERPERM.

**user\_name** A string containing the name for the user ID. Each userid must have a unique name.

**password** The password for the user ID. The password contains the NULL value for the special userids SYS and PUBLIC, preventing anyone from connecting to these user IDs.

**resourceauth (Y/N)** Indicate whether the user has RESOURCE authority. Resource authority is required to create tables.

**dbaauth (Y/N)** Indicate whether the user has DBA (database administrator) authority. DBA authority is very powerful, and should be restricted to as few user IDs as possible for security purposes.

**publishauth (Y/N)** Indicate whether the user has the SQL Remote publisher authority.

**remotedbaauth (Y/N)** Indicate whether the user has the SQL Remote remote DBA authority.

**scheduleauth (Y/N)** Indicate whether the user has SCHEDULE authority. This is currently not used by SQL Anywhere.

**user\_group (Y/N)** Indicate whether the user is a group.

**remarks** A comment string.

When a database is initialized, the following user IDs are created:

- ◆ **SYS** The creator of all the system tables.
- ◆ **PUBLIC** A special user ID used to record PUBLIC permissions.

- ◆ **DBA** The database administrator user ID is the only usable user ID in an initialized system. The initial password is SQL.

There is no way to connect to the SYS or PUBLIC user IDs.

## SYSUSERTYPE system table

```
CREATE TABLE SYS.SYSUSERTYPE (
    type_id SMALLINT NOT NULL,
    creator SMALLINT NOT NULL,
    domain_id SMALLINT NOT NULL,
    nulls CHAR(1) NOT NULL,
    width SMALLINT NOT NULL,
    scale SMALLINT NOT NULL,
    type_name CHAR(128) NOT NULL,
    "default" LONG VARCHAR NULL,
    "check" LONG VARCHAR NULL,
    format_str CHAR(128),
    UNIQUE ( type_name ),
    PRIMARY KEY ( type_id ),
    FOREIGN KEY ( creator )
    REFERENCES SYS.SYSUSERPERM ( user_id ),
    FOREIGN KEY REFERENCES SYS.SYSDOMAIN
)
```

Each row holds a description of a user-defined data type.

**type\_id** A unique identifying number for the user-defined data type.

**creator** The owner of the data type.

**domain\_id** Identifies the data type for the column by the data type number listed in the SYSDOMAIN table.

**nulls** A Y indicates that the user-defined data type does allow nulls. A N indicates that the data type does not allow nulls.

**width** This column contains the length of string columns, the precision of numeric columns, and the number of bytes of storage for all other data types.

**scale** The number of digits after the decimal point for numeric data type columns, and zero for all other data types.

**type\_name** The name for the data type, which must be a valid identifier.

**"default"** The default value for the data type.

**"check"** The CHECK condition for the data type.

**format\_str** Currently unused.

## CHAPTER 49

# SQL Anywhere System Views

### About this chapter

This chapter lists predefined views for the SQL Anywhere system tables.

The system tables described in the chapter "SQL Anywhere System Tables" use numbers to identify tables, user IDs, and so forth. While this is efficient for internal use by SQL Anywhere, it makes these tables difficult for people to interpret. A number of predefined system views are provided that present the information in the system tables in a more readable format.

The definitions for the system views are included with their descriptions. Some of these definitions are complicated, but need not be understood to use the views. They serve as good examples of what can be accomplished using the SELECT command and views.

### Contents

The views are listed alphabetically

## Alphabetical list of views

Each of the views is described by its CREATE statement.

### **SYS.SYSCATALOG**

```
CREATE VIEW SYS.SYSCATALOG ( creator,
                             tname, dbspacename, tabletype, ncols,
                             primary_key, "check", remarks )
AS
SELECT ( SELECT user_name FROM SYS.SYSUSERPERM
          WHERE user_id = SYSTABLE.creator ),
       table_name,
       ( SELECT dbspace_name from SYS.SYSFILE
          WHERE file_id = SYSTABLE.file_id ),
       IF table_type='BASE' THEN 'TABLE'
       ELSE table_type ENDIF,
       ( SELECT count(*) FROM SYS.SYSCOLUMN
          WHERE table_id = SYSTABLE.table_id ),
       IF primary_root = 0 THEN 'N' ELSE 'Y' ENDIF,
       IF table_type <> VIEW' THEN view_def ENDIF,
       remarks
FROM SYS.SYSTABLE
```

Lists all the tables and views from SYSTABLE in a readable format.

### **SYS.SYSCOLAUTH**

```
CREATE VIEW SYS.SYSCOLAUTH ( grantor, grantee,
                             creator, tname, colname )
AS
SELECT ( SELECT user_name FROM SYS.SYSUSERPERM
          WHERE user_id = SYSCOLPERM.grantor ),
       ( SELECT user_name FROM SYS.SYSUSERPERM
          WHERE user_id = SYSCOLPERM.grantee ),
       ( SELECT user_name
          FROM SYS.SYSUSERPERM == SYS.SYSTABLE
          WHERE table_id = SYSCOLPERM.table_id ),
       ( SELECT table_name FROM SYS.SYSTABLE
          WHERE table_id = SYSCOLPERM.table_id ),
       ( SELECT column_name FROM SYS.SYSCOLUMN
          WHERE table_id = SYSCOLPERM.table_id
          AND column_id = SYSCOLPERM.column_id )
FROM SYS.SYSCOLPERM
```

Presents column update permission information in SYSCOLPERM in a more readable format.



## SYS.SYSCOLUMNS

```

CREATE VIEW SYS.SYSCOLUMNS ( creator, cname, tname,
coltype, nulls, length, syslength,
in_primary_key, "colno", default_value, remarks )
AS
SELECT ( SELECT user_name FROM SYS.SYSUSERPERM
        WHERE user_id = SYSTABLE.creator ),
       column_name, table_name,
       ( SELECT domain_name FROM SYS.SYSDOMAIN
         WHERE domain_id = SYSCOLUMN.domain_id ),
       nulls, width, scale, pkey, column_id,
       "default", SYSCOLUMN.remarks
FROM SYS.SYSCOLUMN == SYS.SYSTABLE

```

Presents a readable version of the table SYSCOLUMN. (Note the S at the end of the view name that distinguishes it from the SYSCOLUMN table.)

## SYS.SYSFOREIGNKEYS

```

CREATE VIEW SYS.SYSFOREIGNKEYS ( foreign_creator,
foreign_tname, primary_creator,
primary_tname, role, columns )
AS
SELECT ( SELECT user_name FROM
        SYS.SYSUSERPERM == SYS.SYSTABLE
        WHERE table_id = foreign_table_id ),
       ( SELECT table_name FROM SYS.SYSTABLE
         WHERE table_id = foreign_table_id ),
       ( SELECT user_name
         FROM SYS.SYSUSERPERM == SYS.SYSTABLE
         WHERE table_id = primary_table_id ),
       ( SELECT table_name FROM SYS.SYSTABLE
         WHERE table_id = primary_table_id ), role,
       ( SELECT list( string( FK.column_name,
                            ' IS ', PK.column_name ) )
         FROM SYS.SYSFKCOL KEY JOIN
              SYS.SYSCOLUMN FK, SYS.SYSCOLUMN PK
         WHERE foreign_table_id =
              SYSPFOREIGNKEY.foreign_table_id
         AND foreign_key_id =
              SYSPFOREIGNKEY.foreign_key_id
         AND PK.table_id =
              SYSPFOREIGNKEY.primary_table_id
         AND PK.column_id =
              SYSPFKCOL.primary_column_id )
FROM SYS.SYSFOREIGNKEY

```

Presents foreign key information from SYSPFOREIGNKEY and SYSPFKCOL in a more readable format.

## **SYS.SYSGROUPS**

```
CREATE VIEW SYS.SYSGROUPS ( group_name, member_name
)
AS
SELECT g.user_name, u.user_name
FROM   SYS.SYSGROUP,
        SYS.SYSUSERPERM g,
        SYS.SYSUSERPERM u
WHERE  group_id = g.user_id
AND    group_member = u.user_id
```

Presents group information from SYSGROUP in a more readable format.

## **SYS.SYSINDEXES**

```
CREATE VIEW SYS.SYSINDEXES ( icreator, iname, fname,
creator,
tname, indextype, colnames, interval, level )
AS
SELECT ( SELECT user_name FROM SYS.SYSUSERPERM
WHERE user_id = SYSINDEX.creator ),
index_name,
( SELECT file_name FROM SYS.SYSFILE
WHERE file_id = SYSINDEX.file_id ),
( SELECT user_name FROM SYS.SYSUSERPERM
WHERE user_id = SYSINDEX.creator ),
table_name,
IF "unique" = 'Y' THEN 'Unique'
ELSE 'Non-unique' ENDIF,
( SELECT list( string( column_name,
IF "order" = 'A' THEN 'ASC' i
ELSE 'DESC' ENDIF ) )
FROM SYS.SYSIXCOL == SYS.SYSCOLUMN
WHERE index_id = SYSINDEX.index_id ), 0, 0
FROM SYS.SYSTABLE KEY JOIN SYS.SYSINDEX
```

Presents index information from SYSINDEX and SYSIXCOL in a more readable format.

## **SYS.SYSOPTIONS**

```
CREATE VIEW SYS.SYSOPTIONS ( user_name, "option",
"setting" )
AS
SELECT ( SELECT user_name FROM SYS.SYSUSERPERM
WHERE user_id = SYSOPTION.user_id ),
"option", "setting"
FROM SYS.SYSOPTION
```

Displays option settings contained in the table SYSOPTION in a more readable format.

## SYS.SYSPROCPARMS

```
CREATE VIEW SYS.SYSPROCPARMS ( creator, parmname,
procname,
parmtime, parmmode, parmdomain, length, remarks )
AS
SELECT ( SELECT user_name FROM SYS.SYSUSERPERM
WHERE user_id = SYSPROCEDURE.creator ),
parm_name, proc_name, parm_type,
IF parm_mode_in = 'Y' AND
parm_mode_out = 'N' THEN 'IN'
ELSE IF parm_mode_in = 'N'
AND parm_mode_out = 'Y' THEN 'OUT'
ELSE 'INOUT' ENDIF ENDIF,
( SELECT domain_name FROM SYS.SYSDOMAIN
WHERE domain_id = SYSPROCPARM.domain_id ),
width, SYSPROCPARM.remarks
FROM SYS.SYSPROCPARM == SYS.SYSPROCEDURE
```

Lists all the procedure parameters from SYSPROCPARM in a readable format.

## SYS.SYSREMOTEUERS

```
CREATE VIEW SYS.SYSREMOTEUERS
AS SELECT (SELECT user_name FROM SYS.SYSUSERPERM AS u
WHERE u.user_id=r.user_id) AS user_name,
"consolidate",
(SELECT type_name FROM SYS.SYSREMOTETYPE AS t
WHERE t.type_id=r.type_id) AS type_name,
"address", frequency, send_time,
(IF frequency='A' THEN
NULL
ELSE
IF frequency='P' THEN
IF time_sent IS NULL THEN
current timestamp
ELSE
(SELECT min(minutes(time_sent,
60*hour(a.send_time)
+minute(seconds(a.send_time,59))))
FROM SYS.SYSREMOTEEUSER AS a
WHERE a.frequency='P'
AND a.send_time=r.send_time)
ENDIF
ELSE
IF current date+send_time
```

```
        >COALESCE(time_sent,current timestamp) THEN
        current date+send_time
    ELSE
        current date+send_time+1
    ENDIF
ENDIF
ENDIF) AS next_send,
log_send,time_sent,log_sent,
confirm_sent,send_count,resent_count,
time_received,log_received,confirm_received,
receive_count,rereceive_count
FROM SYS.SYSREMOTEUSER AS r
```

Lists the information in SYSREMOTEUSER in a more readable format.

## **SYS.SYSTABAUTH**

```
CREATE VIEW SYS.SYSTABAUTH ( grantor, grantee,
    screator, stname, tcreator, tname,
    selectauth, insertauth, deleteauth,
    updateauth, updatecols, alterauth, referenceauth
)
AS
SELECT ( SELECT user_name FROM SYS.SYSUSERPERM
    WHERE user_id = SYSTABLEPERM.grantor ),
( SELECT user_name FROM SYS.SYSUSERPERM
    WHERE user_id = SYSTABLEPERM.grantee ),
( SELECT user_name
    FROM SYS.SYSUSERPERM == SYS.SYSTABLE
    WHERE table_id = SYSTABLEPERM.stable_id ),
( SELECT table_name FROM SYS.SYSTABLE
    WHERE table_id = SYSTABLEPERM.stable_id ),
( SELECT user_name FROM
    SYS.SYSUSERPERM == SYS.SYSTABLE
    WHERE table_id = SYSTABLEPERM.ttable_id ),
( SELECT table_name FROM SYS.SYSTABLE
    WHERE table_id = SYSTABLEPERM.ttable_id ),
selectauth, insertauth, deleteauth,
updateauth, updatecols,
alterauth, referenceauth
FROM SYS.SYSTABLEPERM
```

Presents table permission information in SYSTABLEPERM in a more readable format.

## **SYS.SYSTRIGGERS**

```
CREATE VIEW SYS.SYSTRIGGERS ( owner, trigname,
    tname,
    event, trigtime, trigdefn )
AS
```

```

SELECT ( SELECT user_name FROM SYS.SYSUSERPERM
        WHERE user_id = SYSTABLE.creator ),
       trigger_name, table_name,
       IF event = 'I' THEN 'INSERT'
       ELSE IF event = 'U' THEN 'UPDATE'
       ELSE IF event = 'C' THEN 'UPDATE'
       ELSE 'DELETE' ENDIF ENDIF ENDIF,
       IF trigger_time = 'B' THEN 'BEFORE'
       ELSE 'AFTER' ENDIF,
       trigger_defn
FROM SYS.SYSTRIGGER == SYS.SYSTABLE
WHERE foreign_table_id IS NULL

```

Lists all the triggers from SYSTRIGGER in a readable format.

## SYS.SYSUSERAUTH

```

CREATE VIEW SYS.SYSUSERAUTH ( name, password,
                             resourceauth, dbaauth, scheduleauth, user_group )
AS
SELECT user_name, password, resourceauth,
       dbaauth, scheduleauth, user_group
FROM SYS.SYSUSERPERM

```

Displays all the information in the table SYSUSERPERM except for user numbers. Since this view shows passwords, this system view does not have PUBLIC select permission. (All other system views have PUBLIC select permission.)

## SYS.SYSUSERLIST

```

CREATE VIEW SYS.SYSUSERLIST ( name, resourceauth,
                              dbaauth, scheduleauth, user_group )
AS
SELECT user_name, resourceauth,
       dbaauth, scheduleauth, user_group
FROM SYS.SYSUSERPERM

```

Presents all information in SYSUSERAUTH except for passwords.

## SYS.SYSUSEROPTIONS

```

CREATE VIEW SYS.SYSUSEROPTIONS ( "user_name",
                                 "option", "setting" )
AS
SELECT u.name, "option",
       isnull( ( SELECT "setting"
                 FROM sys.sysoptions s

```

```
WHERE s.user_name = u.name
AND s."option" = o."option" ),
"setting" )
FROM SYS.SYSOPTIONS o, SYS.SYSUSERAUTH u
WHERE o.user_name = 'PUBLIC'
```

Display effective permanent option settings for each user. If a user has no setting for an option, this view will display the public setting for the option.

## **SYS.SYSUSERPERMS**

```
CREATE VIEW SYS.SYSUSERPERMS
AS
SELECT user_id, user_name, resourceauth, dbaauth,
       scheduleauth, user_group, remarks
FROM SYS.SYSUSERPERM
```

Contains exactly the same information as the table `SYS.SYSUSERPERM` except the password is omitted. All users have read access to this view, but only the DBA has access to the underlying table (`SYS.SYSUSERPERM`).

## **SYS.SYSVIEWS**

```
CREATE VIEW SYS.SYSVIEWS ( vcreator, viewname,
viewtext )
AS
SELECT user_name, table_name, view_def
FROM SYS.SYSTABLE KEY JOIN SYS.SYSUSERPERM
WHERE table_type = 'VIEW'
```

Lists views along with their definitions.

# Glossary

- About the glossary** This glossary contains words used in the SQL Anywhere User's Guide and industry-wide terms concerning SQL databases and related technologies.
- address** In SQL Remote replication, the destination of replication messages sent by a given message system. Publishers and remote users each have their own address. For a FILE message type, the address is a subdirectory name relative to the directory defined by the SQLREMOTE environment variable. This is typically on a volume that is shared between the replicating databases. SQL Remote stores its messages as replication files in this subdirectory. For a MAPI message type, the address is any valid MAPI address.
- article** In SQL Remote replication, a database object that represents a whole table, or a subset of the rows and columns in a table. Articles are grouped together in publications.
- atomic** A set of operations is atomic if it cannot be broken down into smaller pieces. Transactions are referred to as atomic because they must either be processed entirely or not at all.
- The body of a procedure or trigger can be forced to be atomic by adding the keyword ATOMIC after the BEGIN keyword.
- authority** Determines what structural actions a user can perform in a database. While most users will have no special authorities, a user with DBA authority can grant other users resource authority, DBA authority, or remote DBA authority.

<b>autocommit</b>	An ISQL option. If autocommit is set to TRUE, then a database COMMIT is performed after each successful command and a ROLLBACK after each failed command. The ODBC interface has a setting similar to autocommit. Users of applications communicating with SQL Anywhere through ODBC should check their application documentation for how to set this option. Developers programming directly to the ODBC interface should consult the ODBC documentation for information about implementing autocommit.
<b>backup</b>	It is important to make regular backups of your database files in case of media failure. A backup is a copy of the database file. You can make backups using the SQL Anywhere backup utility or using other archiving software of your choice.
<b>base data type</b>	One of the intrinsic (simple) data types included in SQL Anywhere (such as INTEGER). User-defined data types (including those supplied with SQL Anywhere) are built on base data types.
<b>base table</b>	The tables that permanently hold the data in the database are sometimes called <b>base tables</b> to distinguish them from temporary tables and from views.
<b>batch</b>	A batch is a set of SQL statements sent together to the database engine by an application.
<b>bind variable</b>	In Embedded SQL, a bind variable is a host variable. used to pass values to the database engine.
<b>buffer</b>	An area of memory reserved for some dedicated use, such as storing incoming messages for processing.
<b>business rules</b>	<p>The rules to which the data in a database must conform are often called business rules, as they reflect organizations' operating practices.</p> <p>For example, a business rule limiting customers' credit can be enforced by a trigger in a sales order table. A rule requiring library books to be returned within two weeks can be checked by constraints, so that reminders can be issued or fines calculated.</p>



- cache** To avoid having to access a hard disk every time it needs to retrieve or write information to the database, SQL Anywhere keeps data it may need to access again in the computer's memory, where access is much quicker. The area of memory set aside for this information is called a cache.
- case sensitivity** When creating a database, you can choose whether identifiers in this database (table names, column names, and so on) and values are considered to be case-sensitive in comparisons and string operations.
- If, for example, a database is case-sensitive, emp\_id would not be the same as emp\_ID.
- check constraint** A check constraint allows specified conditions on a column or set of columns in a table to be verified.
- checkpoint** A time at which all dirty pages held in cache are written to disk is called a checkpoint. Once all the dirty pages are written to disk, the checkpoint log is deleted. A checkpoint may occur for one of several reasons, at times specified by the user or decided internally by the database engine.
- checkpoint log** A SQL Anywhere database file is composed of pages. Before a page is updated (made dirty), a copy of the original is always made. The copied pages are the checkpoint log. The checkpoint log is deleted when a checkpoint occurs.
- client** Client is a widely-used term with several meanings. It refers to the user's side of a client/server arrangement: for example, an application that addresses a database, typically held elsewhere on a network, is called a client application.
- client application** In a database context, a client application is any application that communicates with a database engine.
- client/server** A software architecture where one application (the client) obtains information from and sends information to another application (the server).
- In a database context, the server is a database engine, and the client is a database client application. The two applications often reside on different computers on a local area network.

<b>code page</b>	A code page is a character set. SQL Anywhere supports many different national languages and character sets, as long as they are available under the user's operating system.
<b>collation</b>	SQL Anywhere supports many different national languages by providing a choice of collations. Each collation specifies a code page (character set) and a sorting and comparison order for that code page. SQL Anywhere also supports custom collations.
<b>collation sequence</b>	A collation sequence is an ordering of characters used for sorting and comparing strings. Each national language employs a character set or code page, and a collation sequence for that set.
<b>column</b>	All data in relational databases such as SQL Anywhere is held in tables, composed of rows and columns. Each column holds a particular type of information.
<b>command file</b>	A text file containing SQL statements. Command files can be built by yourself (manually) or by database utilities (automatically). The DBUNLOAD utility, for example, creates a command file consisting of the SQL statements necessary to recreate a given database.
<b>command window</b>	The ISQL command window is an edit control for entering SQL statements for execution.
<b>comment</b>	A text description of an object in the database. These remarks are not parsed or executed by the database engine.
<b>commit</b>	When a user sends the SQL command COMMIT, all the work done to that point is applied to the database itself, and can no longer be undone. A commit marks the end of a transaction.
<b>compound statement</b>	A compound statement is a set of SQL statements treated as a unit. The body of a procedure or trigger consists of a compound statement. A compound statement starts with BEGIN and finishes with END.

- compressed database file** A database file that has been compressed to a smaller physical size using SQL Anywhere's database compression utility (DBSHRINK). To make changes to a compressed database file, you must use an associated write file. Compressed database files can be re-expanded into normal database files using SQL Anywhere's database uncompression utility (DBEXPAND).
- concurrency** Multi-user versions of SQL Anywhere support concurrent applications: separate connections which may address the same data in the database, running at the same time. SQL Anywhere provides transaction processing and automatic row-level locking to ensure that information remains consistent and that each concurrent application sees a consistent set of data.
- conflict trigger** In SQL Remote replication, a trigger that is fired when an update conflict is detected, before the update is applied. Specifically, conflict triggers are fired by the failure of values in the VERIFY clause of an UPDATE statement to match the values in the database before the update. They are fired before each row is updated.
- connection** When a client application connects to a database, it specifies several parameters that govern all aspects of the connection once it is established. A user ID, a password, the name of the database to attach to, are all parameters that specify the connection. All exchange of information between the client application and the database to which it is connected is governed by the connection.
- connection ID** A unique number that identifies a given connection between the user and the database.
- You can determine your own connection ID using the following SQL statement inside that connection:
- ```
select connection_property( 'Number' )
```
- connection parameters** When a client application connects to a database, the connection parameters specify the characteristics of the connection. See connection.
- connection string** When a client application connects to a database, it uses connection parameters to specify the characteristics of the connection. These connection parameters are collected together as a connection string.
- See also: connection parameters.

|                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>consolidated database</b> | <p>In SQL Remote replication, a database that serves as the "master" database in the replication setup. The consolidated database contains all of the data to be replicated, while its remote databases may only contain their own subsets of the data. In case of conflict or discrepancy, the consolidated database is considered to have the primary copy of all data.</p>                                                                                                                                                                                                            |
| <b>constraint</b>            | <p>When tables and columns are created they may have constraints assigned to them. A constraint ensures that all entries in the database object, to which it applies satisfy a particular condition. For example, a column may have a UNIQUE constraint, which requires that all values in the column be different. A table may have a foreign key constraint, which specifies how the information in the table relates to that in some other table.</p> <p>See also: integrity, foreign key constraint, primary key constraint, column, table, unique constraint, check constraint.</p> |
| <b>container</b>             | <p>In a graphical user interface, a container is an object that contains other objects. Containers can be expanded by double-clicking them.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>correlated subquery</b>   | <p>A subquery, that contains an outer reference.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>correlation name</b>      | <p>In a SQL SELECT query, a correlation name can be given to a table. The correlation name must then be used to refer to the table in the rest of the query.</p> <p>A correlation name is a convenient way of reducing error and effort in developing long queries that refer to tables with complex names.</p>                                                                                                                                                                                                                                                                          |
| <b>cursor</b>                | <p>A cursor is a handle, or identifier, for a particular SQL query and for the position within the result set that is being accessed. Cursors allow each row of a query that returns more than one row to be processed by a client application individually.</p>                                                                                                                                                                                                                                                                                                                         |
| <b>cursors</b>               | <p>A cursor is a handle, or identifier, for a particular SQL query and for the position within the result set that is being accessed. Cursors allow each row of a query that returns more than one row to be processed by a client application individually.</p>                                                                                                                                                                                                                                                                                                                         |
| <b>cursor stability</b>      | <p>Cursor stability is a concurrency condition, guaranteed by choosing an isolation level of 1, 2, or 3. Cursor stability guarantees that no row fetched through a cursor yields uncommitted data.</p>                                                                                                                                                                                                                                                                                                                                                                                   |

|                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>daemon</b>                 | A background process on a computer, running periodically or all the time, which manages a particular function such as printing services or network communication services.                                                                                                                                                                                                                                                                        |
| <b>data dictionary</b>        | See system tables.                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>data source</b>            | Databases available to ODBC applications are defined through data sources. For a detailed description of ODBC data sources, see "Working with ODBC data sources" in the chapter "Connecting to a Database".                                                                                                                                                                                                                                       |
| <b>data type</b>              | Each <b>column</b> in a table is associated with a particular data type. Integers, character strings, and dates are examples of data types.                                                                                                                                                                                                                                                                                                       |
| <b>database</b>               | A relational database is a collection of tables, related by primary and foreign keys. The tables hold the information in the database, and the tables and keys together define the structure of the database. A database may be stored in one or more database files, on one or more devices.                                                                                                                                                     |
| <b>database administrator</b> | <p>The database administrator (DBA) is a person responsible for maintaining the database. The DBA is generally responsible for all changes to a database schema, and for managing users and user groups.</p> <p>The role of database administrator is built in to SQL Anywhere databases as a user ID. When a database is initialized, a DBA user ID is created. The DBA user ID has authority to carry out any activity within the database.</p> |
| <b>database application</b>   | See client application.                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>database connection</b>    | All exchange of information between client applications and the database takes place in a particular connection. A valid user ID and password are required to establish a connection, and the actions that can be carried out during the connection are defined by the privileges granted to the user ID.                                                                                                                                         |
| <b>database engine</b>        | All access to information in a SQL Anywhere database goes through a SQL Anywhere engine. The specific SQL Anywhere engine you are using will depend on your operating system. Requests for information from a database are sent to the database engine, which carries out the instructions.                                                                                                                                                       |

**database file**

A database is held in one or more distinct database files. The user does not have to be concerned with the organization of a database into files: requests are issued to the database engine about a database, and the engine knows in which file to look for each piece of required information.

Database administrators can create new database files for a database using the "CREATE DBSPACE statement" in the chapter "Watcom-SQL Statements" command.

Each table, together with its associated indexes, must be contained in a single database file.

**database name**

When a database is loaded by an engine, it is assigned a database name. Client applications can connect to a database by specifying its database name.

The default database name is the root of the database file.

**database object**

A database is made up of tables, indexes, views, procedures, and triggers. Each of these is a database object.

**database owner**

The user ID that creates a database is the owner of that database, and has the authority to carry out any changes to that database. The database owner is also referred to as the database administrator, or DBA. A database owner can grant permission to other users to have access to the database and to carry out different operations on the database, such as creating tables or stored procedures.

**datagram**

Communications across a network may take place in a session, (also called a connection, or virtual circuit) or in a connectionless manner. In the connectionless case, the independent packets of information are called datagrams, in analogy with telegrams.

Connectionless communications require routing decisions for all packets, and are not guaranteed. Multi-user editions of SQL Anywhere server use datagrams for their TCP/IP, IPX, and NetDG communication links.

**DBA**

An abbreviation for database administrator, also called the database owner. When a database is first created, using the DBINIT utility, it is created with the single user ID **DBA**, with password **SQL**.

- DBA authority** DBA (DataBase Administrator) authority enables a user to carry out any activity in the database (create tables, change table structures, assign ownership of new objects, create new users, revoke permissions from users, and so on).  
The DBA user has DBA authority by default.
- dbspace** A SQL Anywhere database can be held in multiple files, called dbspaces. The SQL command CREATE DBSPACE adds a new file to the database.  
Each table, together with its associated indexes, must be contained in a single database file.
- DDE** Dynamic data exchange (DDE) is a method for Windows applications to communicate with each other. In any DDE conversation, one application is the client, or destination, while the other application is the server, or source.
- DDE transactions** In DDE, a transaction is an exchange of information between the DDE client and the DDE server. poking, executing., and requesting. are the three types of DDE transaction.
- deadlock** A deadlock occurs when two or more concurrent. transactions become blocked in such a way that they cannot become unblocked. In this situation one of the transactions must roll back its work in order for the other transaction(s) to continue.
- default value** Also known as a "column default" or just "default", this is a value that is automatically assigned to particular columns when a new row is entered into a database table, without any action on the part of the client application, as long as no value is specified by the client application. If the client application does specify a value for the column, it overrides the column's default value. Default values can be user-defined (a string or number) or pre-defined (e.g. a timestamp supplied by the system).
- device** A device is a disk drive, a tape drive, or other information storage medium.
- dirty** A database page is considered dirty when a change is made to it. Before a page is made dirty, a copy of the original page is made and held as the checkpoint log.

**DLL** A dynamic link library, or DLL, is a collection of compiled functions that can be addressed by a running application at run time. DLL's allow a single set of functions to be shared by many applications, and can be updated without updating every application that depends on them. The Windows, Windows NT, and OS/2 operating systems support DLLs.

**domain** Every column is associated with a particular domain: the data type and range of values that constitute valid data for that column.

**driver** A piece of software that manages low-level functions on a computer, such as a communication with a network card. or a printer.

**Dynamic Data Exchange** See DDE.

**Embedded SQL** The native programming interface to SQL Anywhere from C programs. SQL Anywhere embedded SQL is an implementation of the ANSI and IBM standard.

**encryption** Encryption makes it more difficult for someone to decipher the data in your database by using a disk utility to look at the file. File-compression utilities are not able to compress encrypted database files as much as unencrypted ones.

**entity integrity** Each row in every table in a database must be uniquely identifiable in order to be accessed by the database engine. Each row is identified by its primary key. This requirement of identifiability is called entity integrity, and is automatically ensured by SQL Anywhere through a checking of the primary key.

**engine name** When a database engine is started it is assigned an engine name. Client applications specify the engine to which they want to connect by using the engine name.

The default engine name is the first database name.



- environment variable** The DOS and OS/2 operating systems allow users to set environment variables that can be used by applications to identify system-specific information. Windows applications have access to DOS environment variables. SQL Anywhere uses the SQLCONNECT and SQLANY environment variables to identify default connection parameters and home directory for SQL Anywhere.
- entity** In Entity-Relationship design, of databases, a first step is to identify the entities in the information you will incorporate into your database. Entities become tables in the final implementation.
- Entity-Relationship design** Entity-Relationship design is a systematic approach to designing databases, based on a top-down analysis of the tasks you need to perform.
- See also: the chapter "Designing Your Database".
- erase** Erasing a database deletes all tables and data from disk, including the transaction log that records alterations to the database.
- ethernet** Ethernet is a term associated with network cards, a protocol, and a network topology. In a network topology context, an ethernet is commonly associated with a bus network.
- exception handler** In procedures and triggers, an exception handler is defined in the EXCEPTION part of a compound statement, and is code that is executed if an error is encountered in the compound statement.
- exclusive lock** A lock is exclusive when other transactions cannot acquire a lock of similar type on a row. The write lock. is exclusive, while the read lock. is nonexclusive.
- extraction** In SQL Remote replication, the act of synchronizing a remote database with its consolidated database by unloading the appropriate structure and data from the consolidated database, then reloading it into the remote database. Extraction uses direct manipulation of ordinary files—it does not use the SQL Remote message system.

- FILE** In SQL Remote replication, a message system that uses shared files for exchanging replication messages. This is useful for testing and for installations without an explicit message-transport system (such as MAPI).
- foreign key** Tables are related to each other by using foreign keys. A foreign key in one table (the foreign table) contains a value corresponding to the primary key of another table (the primary table). This relates the information in the foreign table to that in the primary table.  
See also: primary key, referential integrity.
- foreign key constraint** A foreign key restricts the values for a set of columns to match the values in a primary key or uniqueness constraint of another table. For example, a foreign key constraint could be used to ensure that a customer number in an invoice table corresponds to a customer number in the customer table. Imposing a foreign key constraint on a set of columns makes that set the foreign key. in a foreign key relationship.  
See also: constraint.
- foreign table** A foreign table is the table containing the foreign key in a foreign key relationship.  
See also: foreign key, primary table. referential integrity.
- forward log** See transaction log.
- full backup** In a full backup, a copy is made of the entire database file itself, and optionally of the transaction log. A full backup contains all the information in the database.
- function** Also called a "user-defined function", this is a type of procedure that returns a single value to the calling environment. A function can be used, subject to permissions, in any place that a built-in non-aggregate function is used.
- grant option** When a user is granted a permission WITH GRANT OPTION, they grant the permission can in turn to other users.
- grantee** A user ID receiving a permission from another user ID is a grantee.

---

|                           |                                                                                                                                                                                                                                                                                                                                                                                                      |
|---------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>grantor</b>            | The user ID granting a permission to another user ID is the grantor.                                                                                                                                                                                                                                                                                                                                 |
| <b>group</b>              | A user group is a database user ID that has been given the permission to have members. User groups are used to make the assignment of database permissions simpler. Rather than assign permissions to each user ID, a user ID is assigned to a particular group, and takes on the permissions assigned to that group.                                                                                |
| <b>handle</b>             | A handle is an integer that uniquely identifies an object. ODBC. makes use of handles to identify environments, connections, and statements.                                                                                                                                                                                                                                                         |
| <b>host variable</b>      | <p>In Embedded SQL, a host variable is a C variable which is identified to the SQL preprocessor. Host variables can be used to send values to the database engine or receive values from the database engine.</p> <p>See also: bind variable.</p>                                                                                                                                                    |
| <b>identifier</b>         | An identifier is any string composed of the characters A through Z, a through z, 0 through 9, underscore (_), at sign (@), number sign (#), or dollar sign (\$). The first character must be a letter. Alternatively, any string of characters can be used as an identifier by enclosing it in double quotes.                                                                                        |
| <b>incremental backup</b> | An incremental backup is a copy of the transaction log. This log contains all the information needed to restore the database to its present state from its state when the transaction log was started.                                                                                                                                                                                               |
| <b>index</b>              | An index on one or more columns of a database table allows fast lookup of the information in these columns, and so can greatly speed up database queries. Specifically, indexes assist WHERE clauses in SELECT statements.                                                                                                                                                                           |
| <b>inner join</b>         | <p>An inner join is one kind of JOIN. operation allowed in the FROM clause of SELECT queries. INNER JOIN is the default type of join.</p> <p>An inner join includes only those rows of the table on each side of the expression that has matching rows in the other table.</p> <p>For a full description, see "FROM clause" in the chapter "Watcom-SQL Statements".</p> <p>See also: outer join.</p> |

- ntegrity** Integrity of information in a database ensures that each row in the database can be uniquely identified (entity integrity) and that relations between rows in different tables are properly maintained (referential integrity). SQL Anywhere provides several tools for maintaining the integrity of information in databases.
- nterprocess  
:ommunication** In operating systems that enable several applications to run at once, interprocess communication allows applications to work together.
- For example, Named pipes. is a method of interprocess communication supported by OS/2 and Windows NT. DDE. is a method of interprocess communication supported by the Microsoft Windows operating environment and the Windows NT operating system.
- PX** IPX is a network-level protocol. by Novell.
- solation levels** The isolation level for instructions within a transaction determines the extent to which other transactions may share the data addressed by the transaction. The isolation level can be set by the user: different isolation levels are appropriate for different kinds of transaction.
- ISQL** ISQL (Interactive SQL) is a SQL Anywhere database administration and browsing utility.
- join** The JOIN clause in a SELECT query enables a single database query to obtain information from several related tables.
- join condition** For any join type. in a join. clause of a SQL query, except for a cross join, a join condition can be specified.
- See also: "FROM clause" in the chapter "Watcom-SQL Statements".
- join type** A JOIN. clause in a SELECT query is one of several join types. The join types supported by SQL Anywhere are cross join, natural join, key join, inner join, and outer join.
- key join** A key join is a join type. used in SELECT queries that take information from more than one table. The key join restricts results to rows where the foreign key value in one table is equal to the corresponding primary key value in the other table.

---

|                           |                                                                                                                                                                                                                                                                                                                                                                                             |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>key</b>                | Database tables may contain two types of key: a <b>primary key</b> , and a <b>foreign key</b> .                                                                                                                                                                                                                                                                                             |
| <b>LAN</b>                | See local area network.                                                                                                                                                                                                                                                                                                                                                                     |
| <b>local area network</b> | <p>A local area network (LAN) is a collection of networked computers characterized by two attributes:</p> <ul style="list-style-type: none"><li>◆ A diameter of not more than a few kilometres.</li><li>◆ Ownership by a single organization.</li></ul> <p>See also: protocol.</p>                                                                                                          |
| <b>locking</b>            | SQL Anywhere places a lock on a row that is being addressed by a transaction. The lock prevents other transactions from having access to the row in a way that could make the data in the database or the data seen by users of client applications inconsistent, while allowing concurrent transactions.                                                                                   |
| <b>log files</b>          | SQL Anywhere maintains a set of three log files to ensure that the data in the database is recoverable in the event of a system or media failure, and to assist database performance.                                                                                                                                                                                                       |
| <b>MAPI</b>               | Microsoft's Message Application Programming Interface, a message system used in several popular e-mail systems such as Microsoft Mail.                                                                                                                                                                                                                                                      |
| <b>media failure</b>      | A media failure occurs when the information on a medium (typically a hard disk drive) becomes unusable. A media failure may occur as a result of damage to the file, the file system, or the actual device. SQL Anywhere includes tools for backup and recovery to minimize the effects of media failure.                                                                                   |
| <b>Message Agent</b>      | In SQL Remote replication, a program (DBREMOTE) that sends and receives replication messages on behalf of a database. A consolidated database's message agent typically runs continuously, receiving replication messages continuously and sending replication messages periodically, while a remote database's agent typically runs on demand, receiving and sending all pending messages. |

|                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>message system</b>   | In SQL Remote replication, a protocol for exchanging messages between the consolidated database and a remote database. SQL Anywhere includes support for a FILE message system (using shared files) and the MAPI message system. In most cases, a consolidated database and a remote database(s) will send and receive messages using the same message system.                                                                                                |
| <b>message type</b>     | In SQL Remote replication, a database object that specifies how remote users communicate with the publisher of a consolidated database. A consolidated database may have several message types defined for it; this allows different remote users to communicate with it using different message systems. A message type is named after a message system (e.g. MAPI), and includes the publisher address for that message system (e.g. a valid MAPI address). |
| <b>messages</b>         | Message based communication between applications or computers does not require a direct connection. Instead, a message sent at one time by an application can be received at another time by another application at a later time.                                                                                                                                                                                                                             |
| <b>named connection</b> | Any connection from ISQL or an embedded SQL application to a database may optionally be given a name. If a client application has several connections in place simultaneously, the user may switch from one connection to another by using the SET CONNECTION command.                                                                                                                                                                                        |
| <b>named pipes</b>      | Named Pipes are an interprocess communication mechanism implemented by a number of leading operating system vendors. Named Pipes are usually implemented atop some transport-level protocol.                                                                                                                                                                                                                                                                  |
| <b>NDIS</b>             | NDIS (Network Driver Interface Specification) is a datalink-level protocol jointly defined by Microsoft and IBM.                                                                                                                                                                                                                                                                                                                                              |
| <b>NetBIOS</b>          | NetBIOS is a transport-level interface defined by IBM.                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>NetBEUI</b>          | NetBEUI is a transport-level protocol.                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>NetWare</b>          | A widely-used network operating system, by Novell. NetWare generally employs the IPX protocol, although the TCP/IP protocol may also be used.                                                                                                                                                                                                                                                                                                                 |

|                                 |                                                                                                                                                                                                                                                                                                           |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>network adapter</b>          | <p>A network adapter is the physical attachment to a computer allowing network communication. Also called a network card.</p> <p>See also: network driver.</p>                                                                                                                                            |
| <b>network architecture</b>     | <p>A network architecture is the physical interconnection of computers on a network. Typical architectures include bus, where all computers connect to a single carrier, and ring, where one computer is directly connected to another to form a closed ring.</p>                                         |
| <b>network card</b>             | <p>A network card is the physical attachment to a computer allowing network communication. Also called a network adapter.</p> <p>See also: network driver.</p>                                                                                                                                            |
| <b>network driver</b>           | <p>A network driver is the software logically located between the operating system and the <b>network card</b>, which allows applications to communicate to the network card. Network cards are usually bundled with a number of network drivers for various operating systems and network protocols.</p> |
| <b>network operating system</b> | <p>An operating system optimized for server deployment, such as Novell NetWare and Microsoft Windows NT Server Edition.</p>                                                                                                                                                                               |
| <b>network server</b>           | <p>A SQL Anywhere database engine that runs on a different PC from the client application (which uses the SQL Anywhere client). The server communicates with the client using a particular network protocol.</p> <p>A network server can support many users connecting from many PCs on the network.</p>  |
| <b>NULL</b>                     | <p>The NULL value is a special value for a database entry, different from any other valid value for any data type. The NULL value represents missing or inapplicable information.</p>                                                                                                                     |
| <b>nullability</b>              | <p>Determines whether a column allows a NULL value to be assigned to it. Columns are typically allowed to be NULL if their values are optional or not always available, and are not required for the data in the database be correct.</p>                                                                 |

|                                   |                                                                                                                                                                                                                                                                                                                                                    |
|-----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>ODBC</b>                       | The Open Database Connectivity (ODBC) interface, defined by Microsoft Corporation, is a standard interface to database management systems in the Windows and Windows NT environments. ODBC is one of several interfaces supported by SQL Anywhere.                                                                                                 |
| <b>ODBC Administrator</b>         | The ODBC. Administrator is a Microsoft program included with SQL Anywhere for setting up ODBC data sources.                                                                                                                                                                                                                                        |
| <b>ODI</b>                        | ODI (Open Datalink Interface) is a data link-level interface defined by Novell.                                                                                                                                                                                                                                                                    |
| <b>Open Database Connectivity</b> | See ODBC.                                                                                                                                                                                                                                                                                                                                          |
| <b>optimizer</b>                  | The SQL Anywhere database engine contains an optimizer which attempts to pick the best strategy for executing each query. The optimizer uses educated guesses about the occurrence of particular elements in the database to select a strategy. The user can provide explicit estimates in order to help tune an execution strategy.               |
| <b>outer join</b>                 | <p>An outer join is one kind of JOIN. operation allowed in the FROM clause of SELECT queries.</p> <p>An outer join may be either a LEFT OUTER or a RIGHT OUTER join. An outer join includes all rows of the table on the LEFT (RIGHT) of the expression whether or not there are matching rows in the other table.</p> <p>See also inner join.</p> |
| <b>outer reference</b>            | A WHERE clause in a subquery. can refer to columns in tables that do not form part of the subquery, but do form part of the main query. This kind of reference is called an outer reference. A subquery that contains an outer reference is called a correlated subquery.                                                                          |
| <b>owner</b>                      | Each object of a database is owned by the user ID that created it. The owner of a database object has rights to do anything with that object.                                                                                                                                                                                                      |
| <b>packet</b>                     | A packet is a communication entity between processes. Messages between two processes are usually fragmented into a number of packets for data transmission from the source and then reassembled at the target host into a single message.                                                                                                          |



|                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>page</b>                   | <p>A SQL Anywhere database file is composed of pages. Before a page is updated (made dirty), a copy of the original is always made in memory. The copied pages are the checkpoint log. The page-size of a database file is specified when the database is created.</p>                                                                                                                                                                                                                                                                                                   |
| <b>page size</b>              | <p>The size of each database page (in bytes). The page size can be 512, 1024, 2048 or 4096 bytes, with 1024 being the default. Other values for the size will be changed to the next larger size. Large databases usually benefit from a larger page size.</p>                                                                                                                                                                                                                                                                                                           |
| <b>passthrough</b>            | <p>In SQL Remote replication, a mode by which the publisher of the consolidated database can directly change remote databases with SQL statements. Passthrough is set up for specific remote users (you can specify all remote users, individual users, or those users who subscribe to given publications). In normal passthrough mode, all database changes made at the consolidated database are passed through to the selected remote databases. In "passthrough only" mode, the changes are made at the remote databases, but not at the consolidated database.</p> |
| <b>password</b>               | <p>Whenever a user connects to a database, a password must be specified. The passwords are stored in the SYS.SYSUSERPERM system table, to which only the DBA has access.</p>                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>performance statistics</b> | <p>Values that reflect the performance of the database system with respect to disk and memory usage. The CURRREAD statistic, for example, represents the number of file reads issued by the engine which have not yet completed.</p>                                                                                                                                                                                                                                                                                                                                     |
| <b>permissions</b>            | <p>Each user has a set of permissions that govern the actions they may take while connected to a database. Permissions are assigned by the DBA or by the owner of a particular database object.</p>                                                                                                                                                                                                                                                                                                                                                                      |
| <b>phantom lock</b>           | <p>A phantom lock is one of three types of lock supported by SQL Anywhere to support concurrency, as part of SQL Anywhere's transaction processing capabilities.</p> <p>A phantom lock is a type of read lock employed at isolation level 3. No other transaction can read or update the row once a phantom lock has been acquired.</p> <p>See also: read lock, write lock, locking.</p>                                                                                                                                                                                 |

- precision** For decimal and numeric data types, the total number of digits (including scale) allowed in the number.
- primary copy** In a replication installation, the primary copy of any data is considered to hold the original of the data. All other places in which the piece of data is held are copies of this original.
- primary table** A primary table is the table containing the primary key in a foreign key relationship.  
See also: primary key, foreign table, referential integrity.
- projection** The result of a SELECT query with DISTINCT specified has all duplicate rows eliminated, compared to the result that would be obtained without the DISTINCT specifier.  
The DISTINCT result is called the projection of the result.
- protocol** The rules and conventions used to communicate between computers are collectively known as a protocol. Instances of protocols include IPX, NetBEUI, and IP. The SQL Anywhere client and server can communicate using a number of different protocols, including NetBIOS, IPX, NamedPipes and TCP/IP.  
See also: IPX, NetBIOS, Named Pipes, TCP/IP.
- protocol stack** Network communications between communications take place according to a set of protocols. These protocols are logically organized in layers, each with a different function. The set of layers forms a protocol stack.
- primary key** Each table in a relational database must be assigned a primary key. The primary key is a column, or set of columns, whose values uniquely identify every row in the table.
- primary key constraint** A primary key constraint identifies one or more columns that uniquely identify each row in a table. Imposing a primary key constraint on a set of columns makes that set the primary key. for the table. The primary key usually identifies the best identifier for a row.  
See also: constraint.

|                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>programming interface</b> | Programs may connect to SQL Anywhere using one of the interfaces supported by SQL Anywhere. Embedded SQL. is SQL Anywhere's native interface. ODBC. is another low-level interface. WSQL DDE. and WSQL HLI. provide higher level interfaces.                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>publication</b>           | In SQL Remote replication, a database object that describes data to be replicated. A publication consists of articles (tables or subsets of tables). Periodically, the changes made to each publication in a database are replicated to all subscribers to that publication as publication updates.                                                                                                                                                                                                                                                                                                                            |
| <b>publication update</b>    | In SQL Remote replication, a periodic batch of changes made to one or more publications in one database. A publication update is sent as part of a replication message to the remote database(s).                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>publisher</b>             | In SQL Remote replication, the single user in a database that can exchange replication messages with other replicating databases.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>remote database</b>       | In SQL Remote replication, a database that exchanges replication messages with a consolidated database. Remote databases may contain all or some of the data in the consolidated database.                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>remote permission</b>     | In SQL Remote replication, the permission to exchange replication messages with the publishing database. Granting remote permissions to a SQL Anywhere user make them a remote user. This requires you to specify a message type, an appropriate remote address, and a replication frequency. In general terms, remote permissions can also refer to any user involved in SQL Remote replication (for example, the consolidated publisher and remote publisher).                                                                                                                                                               |
| <b>qualifier</b>             | <p>The name of a database object, such as a table or a view, may be preceded by a qualifier to indicate its owner and hence uniquely identify the database object. For example, a table named <b>some_table</b> created by a user <b>A_User</b> can be uniquely identified as <b>A_User.some_table</b>.</p> <p>Columns in tables or views may likewise be qualified by the table name (and owner) to identify them uniquely. For example, a column <b>this_column</b> may have the fully qualified name <b>User.some_table.this_column</b>.</p> <p>Use of fully qualified names in SQL queries helps to avoid ambiguities.</p> |

**query**

Information is retrieved from SQL Anywhere databases by submitting a query. A query is an SQL SELECT statement, the clauses of which specify the exact information the database engine must return to the client application.

**read lock**

A read lock is one of three types of lock supported by SQL Anywhere to support concurrency, as part of SQL Anywhere's transaction processing capabilities.

When a transaction reads a row, employing one of the more secure isolation levels available (level 2 or 3), other transactions can read the row, but cannot update it.

See also: phantom lock, write lock, locking.

**referential integrity**

The tables of a relational database are related to each other by foreign keys. SQL Anywhere provides tools that maintain the referential integrity of the database: that is, ensure that the relations between the rows in different tables remain valid.

**relationship**

A step in the Entity-Relationship design of databases, is to identify the relationships between the entities that you have identified.

One-to-many and one-to-one relationships become foreign key relationships in the final implementation, while a many-to-many relationship becomes a table.

**remote DBA authority**

The Message Agent should be run using a user ID with REMOTE DBA authority, to ensure that actions can be carried out, without creating security loopholes.

**remote user**

In SQL Remote replication, a SQL Anywhere user who has been granted remote permissions in a replication setup. When the remote database is extracted from the consolidated database, the remote user becomes the publisher of the remote database, able to exchange publication updates with the consolidated database. While SQL Anywhere groups can also be granted remote permissions, note that users in these "remote groups" do not inherit remote permissions from their group.

|                              |                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>replication</b>           | For databases, a process by which the changes to data in one database (including creation, updating, and deletion of records) are also applied to the corresponding records in other databases. SQL Anywhere supports replication using SQL Remote or Sybase Replication Server.                                                                                                                                                  |
| <b>replication frequency</b> | In SQL Remote replication, a setting for each remote user that determines how often the publisher's message agent should send replication messages to that remote user. The frequency can be specified as on-demand, every given interval, or at a certain time of day.                                                                                                                                                           |
| <b>replication message</b>   | In SQL Remote replication, a discrete communication that is sent from a publishing database to a subscribing database. Messages can contain a mixture of publication updates and passthrough statements (manual SQL statements such as DDL).                                                                                                                                                                                      |
| <b>resource authority</b>    | Resource authority is the permission to create and modify objects of a database schema. Resource authority can be granted only by the DBA.                                                                                                                                                                                                                                                                                        |
| <b>role name</b>             | Each foreign key is assigned a name, called a role name, to distinguish it from other foreign keys in the same table. If no role name is specified by the user, the role name is set to the name of the primary table.                                                                                                                                                                                                            |
| <b>rollback</b>              | When a user sends a rollback SQL statement to the database engine, all work since the last savepoint or since the beginning of the current transaction is undone, and no changes are made to the database by any of the instructions.                                                                                                                                                                                             |
| <b>rollback log</b>          | A log kept in order to cancel changes made to database tables. The rollback log is needed in the event of a ROLLBACK request or a system failure. There is a separate rollback log for each transaction. When a transaction is complete, its rollback log is deleted.                                                                                                                                                             |
| <b>root file</b>             | Unless you specifically create multiple files for your database using the CREATE DBSPACE statement, the database root file is the file your database is held in: the database file.<br><br>If your database is held in multiple database files, the root file is the first file created, which holds the system tables. When supplying the database file parameter in a connection string, you need to supply the root file name. |

- row** All data in relational databases such as SQL Anywhere is held in tables, composed of rows and columns. Each row holds a separate occurrence of each column. In a table of employee information, for example, each row contains information about a particular employee.
- row-level trigger** A trigger that executes BEFORE or AFTER each row modified by the triggering insert, update, or delete operation is changed.
- runtime database engine** The SQL Anywhere Desktop Runtime engine is a royalty-free redistributable database engine. The runtime database engine supports all the database manipulation language features of the full SQL Anywhere, with the exception of procedures and triggers. Also, the runtime database engine does not employ a transaction log.
- savepoint** Within a transaction, a SAVEPOINT statement allows flexibility in committing and rolling back work. Work since the most recent savepoint can be undone using ROLLBACK TO SAVEPOINT. The RELEASE SAVEPOINT disallows any future rollbacks to the most recent savepoint. Before SQL Anywhere version 4.0, savepoints were called subtransactions.
- scale** For decimal and numeric data types, the number of digits allowed after the decimal point.
- schema** The structure of a database is called the schema. The schema is held in the system tables.  
The schema includes the complete definitions of each database object. in the database, including tables, indexes, views, procedures, and triggers.
- search condition** In SQL, a search condition is a WHERE clause in a SELECT statement.
- serializable** A set of concurrent operations is executed in a serializable manner if the net effect of the execution is identical to the result of executing each operation in turn, without any concurrency.  
An isolation level of 3 is needed to guarantee the serializable execution of transactions.

|                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>server</b>   | <p>In SQL Anywhere, servers are database engines—the programs that manage the physical structure of the database and process queries on its data. Servers can be standalone engines or network servers.</p> <p>In Sybase Central, standalone engines and network servers are both called servers.</p>                                                                                                                                                                                      |
| <b>service</b>  | <p>In the Windows NT operating system, applications set up as NT services can run even when the user ID starting them logs off the machine.</p> <p>Running a SQL Anywhere database server as a service under NT allows databases to keep running while not tying up the machine on which they are running.</p>                                                                                                                                                                             |
| <b>session</b>  | <p>Communications across a network may take place in a session, (also called a connection, or virtual circuit) or in a connectionless manner, using a datagram. method. When a connection is established, a route from the source computer to the destination computer is part of the session setup, and routing decisions are not required for every packet.</p> <p>Network server editions of SQL Anywhere use sessions for their NetBIOS and local Named Pipes communication links.</p> |
| <b>size</b>     | <p>For certain data types, the number of characters that the data type can contain.</p>                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>SQL</b>      | <p>Structured Query Language (SQL) is the language used to communicate to SQL Anywhere databases. SQL is very widely used in database applications, and in order to ensure compatibility among databases, SQL is the subject of standards set by several standards bodies.</p>                                                                                                                                                                                                             |
| <b>SQLCA</b>    | <p>A SQL Connection Area (SQLCA) is a segment of database client application code that manages the connection parameters governing the connections between the client application and a database.</p>                                                                                                                                                                                                                                                                                      |
| <b>sqlcode</b>  | <p>A numeric error code identifying database errors.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>sqlstate</b> | <p>A numeric error code identifying database errors. SQL Anywhere provides both the ODBC SQLSTATE, returned to ODBC applications, and the ANSI SQLSTATE, which is returned in the SQLCA to Embedded SQL applications.</p>                                                                                                                                                                                                                                                                  |

|                                  |                                                                                                                                                                                                                                                                                                                                |
|----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>SQLDA</b>                     | The SQL Descriptor Area (SQLDA) is a data structure used for passing dynamic SQL statements to the database engine in Embedded SQL.                                                                                                                                                                                            |
| <b>SQL Remote</b>                | An asynchronous message-based replication system for two-way server-to-laptop, server-to-desktop, and server-to-server replication between SQL Anywhere databases.                                                                                                                                                             |
| <b>standalone engine</b>         | A SQL Anywhere database engine that runs on the same PC as the client application. A standalone engine is typically for a single user on a single PC, but can support several concurrent connections from that user.                                                                                                           |
| <b>statement</b>                 | SQL allows several kinds of statement. Some statements modify the data in a database (commands), others request information from the database (queries), and others modify the database schema itself.                                                                                                                         |
| <b>statement-level trigger</b>   | A trigger that executes after the entire triggering statement is completed.                                                                                                                                                                                                                                                    |
| <b>store-and-forward</b>         | Store-and-forward exchange of information is typical of message based systems, and allows information to be exchanged without a direct connection between applications.                                                                                                                                                        |
| <b>stored procedure</b>          | Stored procedures are procedures kept in the database itself, which can be called from client applications. Stored procedures provide a way of providing uniform access to important functions automatically, as the procedure is held in the database, not in each client application.                                        |
| <b>string</b>                    | In SQL, a string is any sequence of characters enclosed in apostrophes ('single quotes'). To represent an apostrophe inside a string, use a sequence of two apostrophes. To represent a new line character, use a backslash followed by an n (\n). To represent a backslash character, use a sequence of two backslashes (\\). |
| <b>Structured Query Language</b> | See SQL.                                                                                                                                                                                                                                                                                                                       |
| <b>subquery</b>                  | A subquery is a component of a SQL query (SELECT statement) that is itself a query. Subqueries are important tools in constructing complex queries to databases.                                                                                                                                                               |



|                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>submission</b>      | In a SQL Remote installation, changes made to a remote database and sent to the consolidated database are a submission. If and only if the changes are successfully applied at the consolidated database they will be replicated to other databases.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>subscriber</b>      | In SQL Remote replication, a remote user who is subscribed to one or more of a database's publications.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>subscribing</b>     | In a Replication Server or SQL Remote installation, a database that has subscribed to a replication or publication receives updates of changes to the data in that replication or publication.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>subscription</b>    | In SQL Remote replication, a link between a publication and a remote user, allowing the user to exchange updates on that publication with the consolidated database. The user's subscription may include an argument (value) for the publication's SUBSCRIBE BY parameter (if any).                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>subtransaction</b>  | See <b>savepoint</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>synchronization</b> | In SQL Remote replication, the process by which SQL Remote deletes all existing rows from those tables of a remote database that form part of a publication, and copies the publication's entire contents from the consolidated database to the remote database. Synchronization is performed during the initial extraction of the remote database from the consolidated database, and may also be necessary later if a remote database becomes corrupt or gets out of step with the consolidated database (and cannot be repaired using passthrough mode). Synchronization can be accomplished by bulk extraction (the recommended method), by manually loading from files, or by sending synchronization messages through the message system. |
| <b>system failure</b>  | A system failure occurs when a power failure or some other failure causes a computer to fail while there are partially completed transactions. See also: media failure, transaction, backup.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>system catalog</b>  | The system catalog is the list of all tables and views in the database.<br><br>The SQL Anywhere system view SYS.SYSCATALOG presents this information.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

- system object** In a database, a table, view, stored procedure, or user-defined data type that is pre-defined by SQL Anywhere. System tables store information about the database itself, while system views, procedures, and user-defined data types largely support Sybase Transact-SQL compatibility.
- system tables** Every SQL Anywhere database includes a set of tables called the system tables, which hold information about the database structure itself: descriptions of the tables, users and their permissions, and so on.  
The system tables are created and maintained automatically by the database engine. They are owned by the special user ID **SYS**, and cannot be modified by database users.
- system views** Every SQL Anywhere database includes a set of views, which present the information held in the system tables. in a more easily understood format.
- table** All data in relational databases is stored in tables. Each table consists of rows and columns. Each column carries a particular kind of information (a phone number, a name, and so on), while each row specifies a particular entry. Each row in a relational database table must be uniquely identifiable by a primary key.
- TCP/IP** Transmission Control Protocol/Internet Protocol (TCP/IP) is a network protocol supported by SQL Anywhere
- temporary table** Data in a temporary table is held for a single connection only. Global temporary table definitions (but not data) are kept in the database until dropped. Local temporary table definitions and data exist for the duration of a single connection only.
- token ring** A token ring is a specification of a particular network architecture. and protocol. A ring architecture is a topology in which every computer on the ring has a wire in and a wire out and forms a closed loop. A token is passed around the ring to arbitrate between computers that wish to communicate at the same time.
- topics** Each separate panel in an online help system is a topic.
- Transact-SQL** The SQL dialect used in Sybase SQL Server. SQL Anywhere supports a large subset of Transact-SQL.

|                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>transaction</b>            | <p>A transaction is a logical unit of work that should be processed in its entirety by the database (though not necessarily at once) or not at all. SQL Anywhere supports transaction processing, with locking features built in to allow concurrent transactions to access the database without corrupting the data. Transactions begin following a COMMIT or ROLLBACK statement and end either with a COMMIT statement, which makes all the changes to the database required by the transaction permanent, or a ROLLBACK statement, which undoes all the changes made by the transaction.</p> |
| <b>transaction blocking</b>   | <p>A transaction becomes blocked when it must wait for another transaction to finish before it can carry out its task. Proper design of transactions by application developers can minimize the occurrence of transaction blocks, which slow down database operation.</p>                                                                                                                                                                                                                                                                                                                       |
| <b>transaction log</b>        | <p>A log storing all changes made to a database, in the order in which they are made. In the event of a media failure on a database file, the transaction log is essential for database recovery. The transaction log should therefore be kept on a different device from the database files for optimal security.</p>                                                                                                                                                                                                                                                                          |
| <b>transaction log mirror</b> | <p>An identical copy of the transaction log file, maintained at the same time. Every time a database change is written to the transaction log file, it is also written to the transaction log mirror file.</p> <p>A mirror file should be kept on a separate device from the transaction log, so that if either device fails, the other copy of the log keeps the data safe for recovery.</p>                                                                                                                                                                                                   |
| <b>transaction processing</b> | <p>A database engine that supports transaction processing is capable of ensuring that the results of each <b>transaction</b> are either stored in the database in their entirety or not at all. Transaction processing is a key element in promoting secure and reliable databases.</p>                                                                                                                                                                                                                                                                                                         |
| <b>translation driver</b>     | <p>A translation driver. is a part of the data link layer of a protocol stack. Data link layers conform to either ODI. or NDIS. specifications, and translation drivers supplied by networking software vendors enable multiple protocol stacks, requiring both ODI and NDIS drivers, to operate with a single network adapter.</p>                                                                                                                                                                                                                                                             |

|                               |                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>trigger</b>                | A trigger is a procedure stored in the database that is executed automatically by the database engine whenever a particular action occurs, such as a row being updated. Triggers are used to enforce complex forms of referential integrity, or to log activity on database tables.                                                                                                                                |
| <b>two-phase commit</b>       | A mechanism to coordinate transactions across multiple database servers; only needed for distributed databases.                                                                                                                                                                                                                                                                                                    |
| <b>unique constraint</b>      | A unique constraint identifies one or more columns that uniquely identify each row in the table. A table may have several unique constraints.                                                                                                                                                                                                                                                                      |
| <b>unique index</b>           | A unique index on a table is an index for which it is ensured that no two rows have identical values in all columns in the index.                                                                                                                                                                                                                                                                                  |
| <b>unload</b>                 | Unloading a database dumps the structure and/or data of the database to text files (command files for the structure, ASCII comma-delimited files for the data). This may be useful for creating extractions, creating a backup of your database, or building new copies of your database with the same or slightly modified structure. You can also unload the data (but not the structure) of a particular table. |
| <b>updates</b>                | In replication, each set of changes sent from one database to another is an update to a publication or replication.                                                                                                                                                                                                                                                                                                |
| <b>upgrade</b>                | A major release of SQL Anywhere generally means a revised internal database format (to support new database features). When you upgrade your SQL Anywhere software to a major revision (for example, moving from 4.0 to 5.0), you should upgrade your existing databases to the new database format (after making backups of them).                                                                                |
| <b>user-defined data type</b> | A named combination of base data type, default value, check condition, and nullability. Defining similar columns using the same user-defined data type encourages consistency throughout the database.                                                                                                                                                                                                             |
| <b>user account</b>           | Every connection with a database requires a user account. The permissions that a user has are tied to their user account. A user account consists of a user ID and password.                                                                                                                                                                                                                                       |

|                   |                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>user ID</b>    | A string of characters that identifies the user when connecting to a particular database. The user ID, together with a password, constitute a user account.                                                                                                                                                                                                                                                              |
| <b>validate</b>   | When the information in a database, or a database table, is checked for integrity it is validated. See also: entity integrity, referential integrity.                                                                                                                                                                                                                                                                    |
| <b>view</b>       | A view is a computed table. Every time a user uses a view of a particular table, or combination of tables, it is recomputed from the information stored in those tables. Views can be useful for security purposes, and to tailor the appearance of database information to make data access straightforward. As a permanent part of the database schema, a view is a database object. See also: table, database object. |
| <b>VIM</b>        | Vendor-Independent Messaging, a message system used in cc:Mail and Lotus Notes.                                                                                                                                                                                                                                                                                                                                          |
| <b>Watcom-SQL</b> | The SQL dialect used in SQL Anywhere. Watcom-SQL conforms closely to ANSI SQL.                                                                                                                                                                                                                                                                                                                                           |
| <b>Winsock</b>    | Winsock is a specification of the socket library with extensions. The socket library is a collection of functions used to interface to a number of different transport-level protocols. The socket library was initially specified by the University of California at Berkeley for the TCP/IP protocol.                                                                                                                  |
| <b>write file</b> | If a database is used with a write file, all changes made to the database do not modify the database itself, but instead are made to the write file. Write files are useful in applications development, so the developer can have access to the database without interfering with it. Also, write files are used in conjunction with compressed databases and other read-only databases.                                |
| <b>write lock</b> | <p>A write lock is one of three types of lock supported by SQL Anywhere to support concurrency, as part of the SQL Anywhere transaction processing capabilities.</p> <p>When a transaction updates or inserts a row, it acquires a write lock on the row. No other transaction can acquire a lock on the same row.</p>                                                                                                   |
| <b>WSQL DDE</b>   | A high level programming interface to SQL Anywhere for Windows applications.                                                                                                                                                                                                                                                                                                                                             |

**WSQL HLI**

A high level programming interface to SQL Anywhere.

# Index

## Special Characters

- @@char\_convert 557
- @@client\_csid 557
- @@client\_csidname 557
- @@connections 557
- @@cpu\_busy 557
- @@error 557, 904
- @@identity 557, 904
- @@idle 557
- @@io\_busy 557
- @@isolation 557, 904
- @@langid 557
- @@language 557
- @@max\_connections 557
- @@maxcharlen 557
- @@ncharsize 557
- @@nestlevel 557
- @@pack\_received 557
- @@pack\_sent 557
- @@packet\_errors 557
- @@procid 557, 904
- @@rowcount 557, 904
- @@servername 557, 904
- @@spid 557
- @@sqlstatus 557, 904
- @@textsize 557
- @@thresh\_hysteresis 557
- @@timeticks 557
- @@total\_errors 557
- @@total\_read 557
- @@total\_write 557
- @@tranchained 557
- @@trancount 557, 904
- @@transtate 557
- @@version 557, 904

## 2

2000 930

## A

- abort a command 96
- ABS function 938
- ACOS function 938
- ActiveReq 959
- administering SQL Remote 427, 455
- administrator role
  - SQL Anywhere compatibility 540
- ADS
  - support for 660
- aggregate functions 565, 936
  - AVG 122
  - COUNT 122
  - LIST 122
  - MAX 122
  - MIN 122
  - SUM 122
- Alias property 963
- aliases
  - correlation name for tables 115
  - for columns 1142
  - in the DELETE statement 1045
- ALL
  - conditions 147, 578, 912
  - keyword in SELECT statement 1142
- ALL permissions 382
- alloc\_sqllda 709
- alloc\_sqllda\_noinid 709
- allocating disk space 972
- ALLOW\_NULLS\_BY\_DEFAULT option 546, 1157
- alphabetical order 103
- ALTER DBSPACE statement
  - syntax 972
  - TRANSLOG Clause 972
- ALTER permissions 382
- ALTER PROCEDURE statement
  - syntax 974
- ALTER PUBLICATION statement
  - syntax 975
- ALTER REMOTE MESSAGE TYPE statement
  - syntax 976

- ALTER TABLE statement
  - and concurrency 260
  - and foreign keys 241
  - and integrity 229
  - CHECK conditions 235
  - column defaults 230, 231
  - examples 212
  - syntax 976
- ALTER TRIGGER statement
  - syntax 982
- ALTER VIEW statement
  - syntax 983
- altering
  - dbspaces 972
  - publications 975
  - tables 976
- AND conditions 913
- AND keyword 107
- ANSI
  - COMMIT behavior 1152, 1154
  - conformance 893, 1162
  - delete permissions 1159
  - integer overflow behavior 1158
  - NULL behavior 1159
  - update permissions 1159
  - variable behavior 1158
- ANSI\_BLANKS option 1157, 1158
- ANSI\_INTEGER\_OVERFLOW option 1158
- ANSI\_PERMISSIONS option 1159
- ANSINULL option 1159
- ANY
  - conditions 147, 578, 912
- Any outgoing messages will not be identified with
  - a 1135
- apostrophes
  - in SQL 215
  - using 105
- application development systems 26
- architecture
  - about 23, 25, 29, 32
  - basic 25
  - mixed operating systems 32
  - multiuser 29
  - network server 29
  - single-user 25
  - standalone 25
- ARGN function 965
- arithmetic expressions 899
- arithmetic operators 563

- articles
  - system table for 1331, 1332
- ASCII 843
- ASCII file format 1165, 1167
- ASCII function 941
- ASIN function 938
- assertion failed error 1296
- assessing permissions 396
- Async2Read property 955, 959
- AsyncRead property 955, 959
- AsyncWrite property 955, 959
- ATAN function 939
- ATAN2 function 939
- atomic compound statements 285
  - and COMMIT statement 307
  - and EXECUTE IMMEDIATE statement 307
- attributes 182
- audit trail 404
- authorization 663, 893
- auto\_commit
  - ISQL option 1165
- AutoCAD
  - support for 660
- autoincrement 554
  - about 1021
  - default 228, 230, 232
  - maximum value 232
  - primary key values 1063
  - when to use 259
- automatic joins
  - and foreign keys 1311
  - SELECT statement 1142
- AUTOMATIC\_TIMESTAMP option 546, 1157
- Autostop connection parameter 162, 163
- average 936
- AVG function 122, 936

## B

- B+ tree
  - and indexes 327
- BACKGROUND\_PRIORITY option 1150, 1151
- backing up databases 63, 588
- backup functions
  - about 711
- backup utilities
  - options 826
- backups



- about 399, 411, 826
- and SQL Remote 498
- connection parameters 826
- database only 827
- DUMP DATABASE Transact-SQL statement 588
- DUMP TRANSACTION Transact-SQL statement 588
- Embedded SQL functions 711
- for remote databases 504
- for replication 503, 504
- full 411
- live 415, 827
- LOAD DATABASE Transact-SQL statement 588
- LOAD TRANSACTION Transact-SQL statement 588
- online 826
- options 826
- rename and start new transaction log 827
- running database 63
- with Sybase Central 63
- balanced indexes 327
- base tables 184
- batches
  - about 265, 281
  - and control statements 281, 283
  - and data definition statements 281
  - statements allowed in 312
  - Transact-SQL overview 613
  - writing 613
- beep 1165
- BEGIN keyword 283, 614
  - syntax 995
- BEGIN TRANSACTION statement 582
- bell
  - ISQL option 1165
- BETWEEN conditions 110, 577, 909
- binary data
  - GET DATA statement 1081
- BINARY data type 668, 926
  - Transact-SQL 551
- binary data types 926
  - Transact-SQL 551
- binary large objects
  - about 1082
- bind variables
  - about 683
  - DESCRIBE statement 1049

- EXECUTE statement 1062
- OPEN statement 1113
- BIT data type
  - Transact-SQL 551
- bitmaps 200
- bitwise operators 564
- blanks
  - ANSI behavior 1158
- BLOBS 200
  - about 1082
  - GET DATA statement 1082
  - GET DATE statement 1081
  - replication 492
- block fetches 1070
  - OPEN statement 1113
- BlockedOn property 955
- blocking
  - and deadlock 254
  - database option 1151
  - transaction 254
- BLOCKING option 1150
- Borland C++
  - support for 656
- break key 96
- BREAK statement
  - Transact-SQL 622
- browsing databases
  - and isolation levels 256
- B-tree 1338, 1347
- bulk operations 373, 817
- BYTE\_LENGTH function 941
- BYTE\_SUBSTR function 942

## C

- C
  - data types 668
  - interfaces for 653
  - programs 656
- C Set ++ 656
- cache 814, 817
  - and low memory 320
  - and performance 319
  - size 817
- cache buffers 817
- CacheHits property 959
- CacheRead property 955, 959
- CacheReadIndInt property 955, 959

- CacheReadIndLeaf property 955, 959
- CacheReadTable property 955, 959
- CacheWrite property 955, 959
- CALL statement
  - about 265
  - and parameters 288
  - and Transact-SQL 617
  - examples 269
  - syntax 984
- callback function 782
  - in Windows 3.x 716
- capabilities
  - supported 648
- CASCADE 243, 1025
- cascading
  - deletes 243
  - updates 243
- case sensitivity 101, 105, 360, 841, 896, 909, 910
  - and pattern matching 910
  - in the catalog 1339
  - Transact-SQL compatibility 547
- CASE statement 283
  - syntax 986
- CAST function 570, 929, 951
- catalog
  - diagram 1330
  - SQL Server compatibility 540
  - system tables 1329
  - system tables list 1331
- catalog procedures
  - sp\_column\_privileges 602
  - sp\_columns 602
  - sp\_fkeys 602
  - sp\_pkeys 602
  - sp\_special\_columns 602
  - sp\_sproc\_columns 602
  - sp\_stored\_procedures 602
  - sp\_tables 602
  - Transact-SQL 601
- CD-ROM 42
- CEILING function 939
- century issues 930
- CHAINED option 1160
- chained transaction mode 582, 1160
- CHAR data type 918
  - and host variables 1158
  - Transact-SQL 550
- CHAR function 942
- CHARACTER data type 918
  - Transact-SQL 550
- character data types 157
  - Transact-SQL 550
- character sets
  - about 347
  - and ODBC 349
  - Chinese 352
  - choosing 354
  - Japanese 352
  - Korean 352
  - multibyte 352
  - Shift-JIS 352
  - storage 918
  - Taiwanese 352
  - unicode 352
  - UTF8 352
- character strings 894
- CHARACTER VARYING data type 918
- CHECK conditions 235
  - about 225, 227, 1025
  - deleting 238
  - modifying 238
  - on columns 228, 235
  - on tables 228, 237
  - Transact-SQL 585
  - user-defined data types 236
- CHECK ON COMMIT clause
  - and referential integrity 1025
- CHECKPOINT
  - checkpoint log 988
  - checkpoint log 401
  - CHECKPOINT statement
    - and recovery 401
    - syntax 988
  - CHECKPOINT\_TIME option 1150, 1152
- checkpoints
  - CHECKPOINT\_TIME option 1152
  - scheduling of 402
  - urgency of 402
- Chinese character set 352
- Chkpt property 959
- ChkptFlush property 959
- ChkptPage property 959
- CLEAR ISQL command 848
- Client
  - defined 31
  - SQL Anywhere 29
- client applications

- about 25, 30
- and databases 166
- SQL Anywhere programs 36, 40
- Sybase Central 39
- using 26
- CLOSE statement
  - about 678
  - in procedures 295
  - syntax 989
- CLOSE\_ON\_ENDTRANS option 1160
- COALESCE function 965
- code editor 57
- code pages 354, 843
  - and data storage 918
- col\_length SQL Server function 573
- col\_name SQL Server function 573
- cold links 767
- collation file format 357
- collation sequences 830, 843
- collations 354
  - about 347
  - and pattern matching 910
  - collation file format 357
  - custom 357, 358
  - file format 358
  - multibyte 352
- column defaults
  - about 231
- column names
  - long 1051
- columns 259
  - about 101, 182
  - adding 213
  - adding using Sybase Central 52
  - aliases 1142
  - altering constraints in Sybase Central 237
  - altering defaults in Sybase Central 231
  - altering definition 978
  - and user-defined data types 236, 928
  - changing 213
  - changing heading name 1142
  - CHECK conditions 228, 235, 585
  - choosing data types for 199
  - choosing names 199
  - constraints 200, 227, 228, 235, 928, 1023
  - creating 52
  - creating constraints in Sybase Central 237
  - creating defaults in Sybase Central 231
  - defaults 227, 230, 231, 259, 585
  - deleting 213
  - deleting constraints in Sybase Central 237
  - designing 199
  - drag and drop 54
  - dropping defaults in Sybase Central 231
  - identity 554
  - in publications 471
  - in the system tables 1334
  - list of 78, 95
  - making a primary key with Sybase Central 53
  - name 691
  - naming 215, 897, 901
  - ordering 104
  - permissions on 1333
  - renaming 213, 980
  - rules 585
  - selecting from a table 104
  - shortening 213
  - SYSCOLUMNS system view 1357
  - timestamp 552
- comma delimited files 1165, 1167
- command delimiter
  - setting 309, 1165
- command echo 1165
- command files
  - about 1126
  - and ISQL 205
  - building 152
  - Command window 152
  - overview 152
  - parameters 154, 1126
- command line summary 808
- command line utilities and Unix 809
- command recall 94
- Command window 87
  - ISQL 72
- command-line switches
  - about 816
  - and services 520
  - for server 816
  - in configuration file 816
- commands
  - editing in ISQL 76
  - executing in ISQL 72
  - getting 152
  - getting in ISQL 72
  - interrupting in ISQL 79
  - loading 152
  - loading in ISQL 72

- previous 76
- recalling in ISQL 76
- saving 153
- saving in ISQL 72
- stopping in ISQL 79
- COMMENT statement
  - syntax 991
- comments
  - about 153, 915
  - comment indicators 915
  - indicators 580
- Commit property 955
- COMMIT statement
  - about 131
  - and procedures 308
  - and transactions 246
  - in compound statements 285
  - syntax 993
  - Transact-SQL 584
- COMMIT\_ON\_EXIT
  - ISQL option 1165
- CommitFile property 959
- commits
  - COOPERATIVE\_COMMIT\_TIMEOUT option 1152
  - COOPERATIVE\_COMMITS option 1152
  - CREATE TABLE statement 212
  - DELAYED\_COMMIT\_TIMEOUT option 1154
  - DELAYED\_COMMITS option 1154
  - DROP TABLE statement 214
- CommLink property 955
- CompanyName property 959
- comparing dates and times 925
- comparison conditions
  - about 908
  - Transact-SQL 576
- comparison operators 577, 908
- comparisons
  - about 105, 106, 907
  - using subqueries 145
- compatibility
  - of SQL Anywhere and SQL Server databases 544
  - QUERY\_PLAN\_ON\_OPEN 1161
- compatibility of SQL 606
- compile and link process 657, 658
- compilers supported 656
- components 808
- compound search conditions 107
- compound statements 283, 614
  - about 995
  - and COMMIT statement 285
  - and ROLLBACK statement 285
  - atomic statements 285
  - declarations in 284
  - SQL statements allowed in 286
  - using 283
- compressed databases 874
- COMPUTE clause
  - Transact-SQL 593
- concatenating strings 563, 905
- concurrency 248
  - about 247, 248, 259
  - and data definition 259
  - and data definition statements 260
  - and locks 248
  - and performance 249
  - and primary keys 259
  - consistency 248
  - inconsistency 248
- conditions 108
  - about 907
  - and GROUP BY clause 125
  - search 105, 106, 110
- configuration
  - file, for database engine 816
- CONFIGURE statement
  - syntax 998
- configuring an Open Server 642
- conflict resolution in SQL Remote 507
- conflicts
  - cyclical blocking 255
  - locking 254
  - reporting in SQL Remote 507
  - resolution of 263
  - resolving in SQL Remote 507, 509, 510
  - VERIFY\_ALL\_COLUMNS option 510
- ConnCount property 963
- connect
  - granting 394
  - permission 380
- CONNECT statement 733
  - and Embedded SQL 662
  - example 140
  - syntax 999
- connecting 84
  - to a database 70, 161

- Connecting to a database
  - using Sybase Central 48
  - using the default password 48
  - using the default user ID 48
- connection properties
  - displaying 1321
- connection\_property function 953
- connections
  - about 162
  - dropping 820
  - parameters 162, 163, 166, 173
  - properties of 955
  - same machine 26
  - string 162
- consistency 248
  - during transactions 248
- consolidate permissions 1089, 1134
  - granting 466
  - revoking 466
- consolidated database 1134
  - publishing 1135
- consolidated databases 422
  - setting up 433, 442
- constant expressions
  - defaults 234
- constants
  - in expressions 900
  - Transact-SQL 561
- constraints 1023
  - about 225, 227, 978
- containers 49
- CONTINUE statement
  - Transact-SQL 622
- ContReq property 959
- control statements 606, 614
  - about 265, 995
  - BEGIN keyword 283
  - CALL statement 984
  - CASE statement 283, 986
  - compound statements 283
  - END keyword 283
  - IF statement 283, 1095
  - LEAVE statement 1104
  - LOOP statement 283, 1108
  - Transact-SQL BREAK statement 622
  - Transact-SQL CONTINUE statement 622
  - Transact-SQL GOTO statement 618
  - Transact-SQL IF statement 618
  - Transact-SQL RETURN statement 621
  - Transact-SQL WHILE statement 622
  - WHILE statement 283, 1108
- conventions
  - documentation 9
- conversion errors 372
- CONVERSION\_ERROR option 1157
- CONVERT function 570, 951
- converting ambiguous strings 932
- converting data types 951
- COOPERATIVE\_COMMIT\_TIMEOUT option 1150, 1152
- COOPERATIVE\_COMMITS option 1150, 1152
- copyright
  - retrieving 1327
- correlated subqueries
  - defined 148
- correlation names
  - about 115
  - defined 148
  - in the DELETE statement 1045
- COS function 939
- COT function 939
- COUNT function 121, 122, 123, 218, 936
- CREATE DATABASE statement
  - Transact-SQL 588
- CREATE DATATYPE statement
  - syntax 1002
- CREATE DBSPACE statement
  - about 207
  - preallocating space 209
  - syntax 1004
- CREATE DEFAULT statement
  - Transact-SQL 585
- CREATE DOMAIN statement
  - syntax 1002
- CREATE FUNCTION statement
  - about 273
  - syntax 1005
- CREATE INDEX statement
  - and concurrency 260
  - and table use 1008
  - example 326
  - syntax 1007
  - Transact-SQL 585
- CREATE MESSAGE statement
  - Transact-SQL 586
- CREATE PROCEDURE statement
  - examples 268
  - parameters 287



- EXPLAIN statement 1066
  - fetching 1068
  - in Embedded SQL 678
  - in ODBC 756
  - in procedures 295, 297
  - in triggers 295
  - looping over 1073
  - on SELECT statements 297
- OPEN statement 1112
  - opening 1161
- WITH HOLD clause 1112
- Cursors property 955
- curunreservedpgs SQL Server function 573
- custom collations 358

## D

### data

- and locks 248
- consistency 248
- deleting 227
- duplicated 226, 227
- exporting 368
- invalid 226, 227
- loading large amounts 320
- modifying 227
- updating 227

### data definition

- and concurrency 259

- data definition statements
  - and concurrency 260

### data entry

- and isolation levels 256

- data integrity 244

- data normalization 191

- data sources 169

- about 173, 174, 179

- adding 174

- connection keywords 163

- connection overview 162

- modifying 179

- removing 179

- working with 173

- data type conversion functions 570

- data types 684, 691

- about 182, 917

- and OmniCONNECT support 650

- binary 551, 926

- bit 551

- C data types 668

- character 550, 918

- choosing 199

- converting 570, 929, 951

- creating 1002

- data and time 552

- date 922

- date and time 922

- decimal 550

- defined 897

- dropping user-defined 1053

- in the system tables 1335, 1353

- integer 549

- money 551

- numeric 920

- time 922

- timestamp 552

- Transact-SQL 549, 550, 551, 552

- user-defined 555, 927, 1002, 1353

- data\_pgs SQL Server function 573

- database containers

- expanding with Sybase Central 49

- database design

- about 191

- and performance 320

- normalizing data 191

- verifying 197

- verifying a design 197

- database engines

- about 26, 36, 42, 263, 814

- and portable computers 263

- and updating databases 263

- command line 814

- connection 27

- deployment 263

- operating systems 30

- runtime edition 37

- starting 1173

- stopping 42, 1176

- unlimited runtime license 263

- v command line switch 263

- Database extraction utilities 859

- database files

- adding 207

- allocating space for 209

- creating 1004

- dbspaces 1004

- erasing 835

- storing indexes in 1007
- uncompressing 874
- database information 838
- database management
  - using Sybase Central 39, 45
- database objects
  - commenting 991
- database options
  - ALLOW\_NULLS\_BY\_DEFAULT 546
  - AUTOMATIC\_TIMESTAMP 546
  - DATE\_ORDER 923
  - QUOTED\_IDENTIFIER 546, 562
  - set by Open Server Gateway 644
- database pages
  - caching 320
  - displaying size of 837
- database properties
  - changing 45
  - displaying 1321
- database schema
  - about 1329
  - viewing with Sybase Central 49
- database security 524
- database servers
  - about 36
  - operating systems 30
- database threads
  - blocked 255
- database utilities
  - and database connections 168
  - DBBACKUP 42
  - DBERASE 42
  - DBINFO 42
  - DBINIT 42
  - DBLOG 42
  - DBSHRINK 42
  - DBSTOP 42
  - DBUNLOAD 42
  - DBUPGRAD 42
  - DBVALID 42
  - DBWRITE 42
  - ISQL 40
  - log translation 404
  - REBUILD 42
  - Wizards 63
- DatabaseFile connection parameter 162, 163
- DatabaseName connection parameter 162, 163
- databases
  - about 42
  - administering 45
  - administrator, defined 377
  - alias 31
  - and portable computers 262
  - and write files 263
  - backing up 42, 63
  - building 203
  - cache size 814
  - checking validity of 42
  - compacting 42
  - compressed 42, 874
  - compressing 263
  - configuring for Transact-SQL 544
  - conflict resolution 263
  - connecting to 48, 161, 162, 166
  - consolidated 422
  - copying 262
  - creating 42, 203, 206, 841
  - creating Transact-SQL-compatible 544
  - data integrity 225, 226
  - designing 181, 185, 187
  - devices 27
  - erasing 42, 209, 835
  - files 27, 31, 1336
  - ignoring trailing blanks 544
  - information 837, 838
  - initializing 42, 206, 841
  - initializing for Transact-SQL 544
  - integrity constraints 227
  - large 263
  - library memory 718
  - loading data into 1105
  - maintaining 42
  - managing 206
  - managing 45
  - managing using Sybase Central 39
  - memory 719
  - multiple 27, 30
  - multiple-file 28, 30, 207, 209
  - name 31
  - objects 184
  - page usage 837
  - permissions 376
  - planning 185
  - problems updating 262
  - properties of 963
  - read-only 42
  - rebuilding 42, 857
  - relational 182



- remote 422, 426, 438, 445
- schema 1329
- SQL Anywhere 27
- starting 814, 816, 1172
- stopping 867, 1175
- storage on disk 209
- structure 1329
- synchronizing 445, 446
- system procedures 1319
- system tables 1329
- Transact-SQL compatibility 544
- Transact-SQL CREATE DATABASE 588
- Transact-SQL DROP DATABASE 588
- Transact-SQL DUMP DATABASE 588
- Transact-SQL LOAD DATABASE 588
- unloading 42, 878
- unloading tables from 1182
- updates 263
- updating 262
- updating from a transaction log 262, 263
- upgrading 7, 42, 882, 883
- using ISQL to manage 205
- using Sybase Central 204
- validating 411, 886
- working with compressed 263
- write files 890
- DATALENGTH function 954
- datalength SQL Server function 573
- DataSourcename connection parameter 162, 163
- date and time data types
  - Transact-SQL 552
- date and time functions 568
  - Transact-SQL 568
- DATE data type 922, 946
  - Transact-SQL 552
- date format
  - database option 1152
- DATE function 946
- date order
  - database option 1153
- date to string conversions 933
- DATE\_FORMAT option 1150
- DATE\_ORDER option 923, 1150
  - and ODBC 923
- dateadd Transact-SQL function 568
- DATEFORMAT function 946
- DATENAME function 947
- datetime Transact-SQL function 568
- datepart Transact-SQL function 568
- dates 106, 110, 157, 922, 946, 947, 948, 949, 950
  - ambiguous string conversions 932, 933
  - combining 107
  - comparing 925
  - compound 107
  - conversion problems 933
  - formatting 946
  - interpreting strings as dates 923
  - unambiguous specification of 923
  - year 2000 930
- DATETIME data type 668
  - Transact-SQL 552
- DATETIME function 946
- DAY function 946
- DAYNAME function 947
- DAYS function 947
- db\_abort\_request 715
- db\_backup 711
- db\_break\_handler 721
- db\_build\_parms 708
- db\_cancel\_request 715
- db\_catch\_break 721
- db\_catch\_critical 721
- db\_delete\_file 714
- db\_destroy\_parms 708
- db\_find\_engine 707
- db\_fini 702
- db\_finished\_request 720
- db\_free\_parms 708
- db\_get\_sqlca 718
- DB\_ID function 954
- db\_jd SQL Server function 573
- db\_init 702
- db\_is\_working 715
- DB\_NAME function 954
- db\_name SQL Server function 573
- db\_parms\_connect 708
- db\_parms\_disconnect 708
- db\_process\_a\_message 717
- db\_property function 954
- db\_register\_a\_callback 716
- db\_release\_break 721
- db\_release\_critical 721
- db\_sending\_request 720
- db\_set\_sqlca 718
- db\_start 708
- db\_start\_database 706
- db\_start\_engine 705

- db\_stop 709
- db\_stop\_database 706
- db\_stop\_engine 707
- db\_string\_connect 704
- db\_string\_disconnect 704
- db\_working 715
- DBA
  - defined 377
- DBA authority
  - about 376
  - and permissions 386
  - granting 381
  - in the system tables 1352
  - not inheritable 386
- DBAlloc 718
- dBASE file format 1165, 1167
- DBBACKUP 42, 411, 826
  - command line 826
- DBCLIENT 29, 32, 37
- DBCLIENW 29
- DBCOLLAT 357, 358, 830
- DBENG50S 37
- DBENG50W 37
- DBERASE 42, 209, 835
- DBEXPAND 874
- DBFree 719
- DBINFO 42, 320, 838
- DBINIT 42, 206, 320, 360, 841
  - ignoring trailing blanks 544
- DBL50T.DLL 172
- DBL50W.DLL 172
- DBLOG
  - command line 871
- DBNumber property 955
- dbo user 545, 598
  - system objects 858, 876
- DBOS50 643, 853
- DBOSINFO 855
- DBOSSTOP 856
- DBRealloc 719
- dbremote 447
  - about 496, 863
  - and security 497
  - command line 863
- DBSHRINK 42, 263, 833
- DBSPACE 1336
- dbspaces
  - altering 972
  - creating 207, 1004
  - dropping 1053
  - managing 589
  - preallocating space 209
- DBSTOP 42, 867
- DBTOOL statement
  - creating databases 207
  - syntax 1035
- DBTRAN 262, 263, 404, 418
  - command line 850
- DBUNLOAD 42, 369, 878
  - and replication 505
- DBUPGRAD 7, 42, 883
- DBVALID 42, 411, 886
- DBWATCH
  - and services 531
- DBWRITE 42, 263, 890
- DBXTRACT 445, 484
  - about 859
  - command line 859
- DDE
  - support for 654
- deadlock
  - and transaction blocking 254
- deadlocks
  - reasons for 255
- DECIMAL data type 668, 920
  - Transact-SQL 550
- decimal data types
  - Transact-SQL 550
- decimal precision
  - database option 1155
- decision support
  - and isolation levels 256
- DECL\_BINARY 668
- DECL\_DECIMAL 668
- DECL\_FIXCHAR 668
- DECL\_VARCHAR 668
- declaration section 1038
  - about 667
- declarations 284
- DECLARE CURSOR statement
  - syntax 1039
- DECLARE statement
  - about 678
  - and compound statements 614
  - in procedures 295, 301
  - syntax 995
  - Transact-SQL compatibility 614
- DECLARE TEMPORARY TABLE statement

- syntax 1043
- defaults
  - about 227
  - and foreign keys 242
  - autoincrement 228, 232, 1021
  - column 230
  - column, and Sybase Central 231
  - constant expressions 234
  - creating 230
  - current date 227, 231
  - current time 231
  - current timestamp 231
  - NULL 233
  - string and number 233
  - Transact-SQL 585
  - user 1021
  - user ID 232
  - with transactions and locks 259
- DEGREES function 939
- DELAYED\_COMMIT\_TIMEOUT option 1150, 1154
- DELAYED\_COMMITS option 1154
- DELETE permissions 382
- DELETE positioned statement
  - syntax 1047
- DELETE statement
  - about 132
  - and errors 134
  - and integrity 227, 243
  - examples 134
  - syntax 1045
  - Transact-SQL 589
- delete tables 56
- DELETE\_OLD\_LOGS
  - replication option 1163
- DELETE\_OLD\_LOGS option 504
  - resetting truncation offset 871
  - transaction log options 871
- deletes
  - ANSI behavior 1159
  - Transact-SQL permissions 1159
- deleting
  - services 521
- deleting all rows from a table 1180
- dependencies
  - of services 528, 529
- deployment 170, 262
  - on laptop computers 262, 263
  - SQL Remote 494
- DESCRIBE statement 685
  - long column names 1051
  - syntax 1049
- descriptor
  - EXECUTE statement 1062
  - FETCH statement 1068
  - OPEN statement 1112
  - PREPARE statement 1120
  - PUT statement 1124
  - UPDATE (positioned) statement 1186
- designing publications 475, 476, 477, 478, 479
- device errors 719
- devices
  - managing 589
  - recovery from failure of 416
- DIF file format 1167
- DIFFERENCE function 942
- digits
  - maximum number 1155
- dirty reads 248, 251
- DISCONNECT statement
  - syntax 1052
- disk full
  - error writing to transaction log 406
- DiskRead property 955, 959
- DiskReadIndInt property 955, 959
- DiskReadIndLeaf property 955, 959
- DiskReadTable property 955, 959
- disks
  - access 817
  - fragmentation and performance 320
  - managing 209, 320, 589
  - recovery from failure of 416
- DiskSyncRead property 955, 959
- DiskSyncWrite property 955, 959
- DiskWaitRead property 955, 959
- DiskWaitWrite property 955, 959
- DiskWrite property 955, 959
- DISTINCT keyword 1142
- distributed database systems 260
- divide by zero
  - error 1160
- DIVIDE\_BY\_ZERO\_ERROR option 1160
- DLL entry points 701
- DLLs
  - calling from functions 314
  - calling from procedures 314
- documentation
  - about 9

- conventions 9
- formats 9
- domains *See* user-defined data types
- DOS
  - and SQL Remote 457
  - ISQL 81
  - ISQL Statistics window 323
  - memory allocatoin 718
- DOUBLE data type 920
  - Transact-SQL 550
- DOW function 947
- DOWN ISQL command 848
- drag and drop
  - columns 54
- DROP CONNECTION statement
  - syntax 1055
- DROP DATABASE
  - Transact-SQL 588
- DROP DATATYPE statement
  - syntax 1053
- DROP DBSPACE statement
  - syntax 1053
- DROP DOMAIN statement
  - syntax 1053
- DROP FUNCTION statement
  - syntax 1053
- DROP INDEX statement
  - syntax 1053
- DROP OPTIMIZER STATISTICS statement
  - about 324
  - syntax 1056
- DROP PROCEDURE statement
  - syntax 1053
- DROP PUBLICATION 473
- DROP PUBLICATION statement
  - syntax 1057
- DROP REMOTE MESSAGE TYPE statement
  - syntax 1058
- DROP statement
  - and concurrency 260
  - dropping indexes 224
  - syntax 1053
- DROP STATEMENT statement
  - syntax 1059
- DROP SUBSCRIPTION statement
  - syntax 1061
- DROP TABLE statement
  - example 214
  - syntax 1053

- DROP TRIGGER statement
  - about 279
  - syntax 1053
- DROP VARIABLE statement
  - syntax 1060
- DROP VIEW statement
  - about 139
  - example 221
  - syntax 1053
- dropping
  - databases 588
  - indexes in Sybase Central 224
  - publications 1057
  - subscriptions 1061
  - users 1133
  - views in Sybase Central 222
- dropping publications 473
- dropping users
  - consolidate 1134
- DT\_STRING 710
- DUMMY table 1331
- DUMP DATABASE statement
  - Transact-SQL 588
- DUMP TRANSACTION statement
  - Transact-SQL 588
- duplicate data
  - as invalid data 226
- dynamic cursors
  - example 738
- DYNAMIC SCROLL cursors 1039
- dynamic SQL 682

## E

- EBCDIC 843
- echo
  - ISQL option 1165
- editing commands 94, 105
- editor 91
- ELSE
  - IF expression 906
- e-mail 423
  - MAPI link 429
  - SMTP link 429
  - system procedures 1323, 1324, 1325
  - VIM link 429
- Embedded SQL
  - about 653, 656

- authorization 893
- character strings 894
- command summary 727
- compile and link process 657, 658
- cursors 678, 730
- dynamic 682
- dynamic cursors 738
- fetching data 677
- functions 701, 718
- host variables 667
- import libraries 660
- interface 34
- line numbers 893
- memory usage 718
- static 682
- encryption 841
- END keyword 283, 614
  - syntax 995
- ENDIF
  - IF expression 906
- engine name
  - defined 31
- engine properties
  - displaying 1321
- EngineName connection parameter 162
- engines
  - properties of 959
- entities
  - about 185
- entity integrity 1310
  - about 228
  - enforcing 239
- Entity-Relationship design
  - overview 185
- environment variables 168, 808, 810
  - Data Sources 169
  - SQLANY 810
  - SQLCONNECT 168, 810
  - SQLREMOTE 811
  - SQLSTART 811
  - TMP 812
- erasing databases 209, 835
- error
  - buffer 674
  - messages
    - Embedded SQL function 714
    - strings 674
- errors
  - codes 1191, 1201
  - conversion 372
  - divide by zero 1160
  - error messages 1210
  - handling in ISQL 1167
  - in procedures 300
  - in Transact-SQL 620
  - in triggers 300
  - messages 1191
  - RAISERROR statement 620
  - reporting in SQL Remote 507
  - SIGNAL statement 1171
  - SQLSTATE values 1201
  - user-defined messages 1351
- escape character
  - INPUT statement 1098
  - OUTPUT statement 1115
- ESTIMATE function 965
- ESTIMATE\_SOURCE function 966
- estimates
  - for search conditions 579
  - in queries 335
  - providing 334
  - row-count estimates 1155
- exception handlers
  - declaring 284
  - using in procedures and triggers 304
- EXCEPTION statement
  - syntax 995
- exceptions
  - declaring 301
- exclusive locks 250
- EXEC SQL
  - Embedded SQL 662
- executable files 808
- EXECUTE IMMEDIATE statement
  - and atomic compound statements 307
  - in procedures 307
  - syntax 1064
- EXECUTE statement 683
  - syntax 1062
  - Transact-SQL 617
- executing commands 92
- execution plan
  - ISQL statistics window 324
- EXISTS conditions 578, 912
- EXIT statement
  - syntax 1065
- EXP function 939

- expanding
  - viewing 49
- EXPERIENCE\_ESTIMATE function 966
- EXPLAIN statement
  - syntax 1066
- exporting data 364, 366, 1115, 1167
  - performance 373
  - SELECT statement 1141
- expressions 899
  - Transact-SQL 561
- ExtendDBWrite property 959
- ExtendTempWrite property 959
- external procedures 314
- extraction utility 484

## F

- failure
  - recovery from 399
- FALSE condition 913
- far pointers
  - and Embedded SQL 701
- Feb 29 932
- FETCH
  - multi-row 686
  - wide 686
- FETCH statement 685
  - about 678
  - in procedures 295
  - syntax 1068
- fetching data
  - in Embedded SQL 677
- file management 320
- FILE message type
  - about 976
- FILE messages
  - about 1015, 1058
- File property 963
- file system 817
- files
  - dbspaces 1004
- FileVersion property 963
- fill\_s\_sqlda 710
- fill\_sqlda 709
- FIPS
  - conformance 893, 1162
- FIRE\_TRIGGERS option 1157, 1160
- FIXCHAR data type 668
- FIXED file format 1165, 1167
- FLOAT data type 920
  - Transact-SQL 550
- FLOOR function 939
- FOR BROWSE clause
  - Transact-SQL 593
- FOR READ ONLY clause
  - Transact-SQL 593
- FOR statement
  - in procedures 299
  - syntax 1073
- FOR UPDATE clause 1070
  - Transact-SQL 593
- foreign keys
  - about 183
  - and inserts 133
  - and integrity 241, 1310
  - and performance 325
  - and query execution 328
  - and referential integrity 241
  - and tables 117
  - as invalid data 226
  - choosing 201
  - defined 183
  - in the system tables 1336, 1337, 1338
  - integrity constraints 1024
  - mandatory 241
  - optional 241
  - role names 1024
  - system views 1357
  - unnamed 1024
  - using to improve performance 322
- foreign table
  - in the system tables 1337
- format 1165
- forward log 403
- FoxPro file format 1165, 1167
- fragmentation
  - and performance 319
- free\_filled\_sqlda 710
- free\_sqlda 710
- free\_sqlda\_noind 710
- FreeWriteCurr property 959
- FreeWritePush property 959
- frequency
  - of sending messages 1089, 1092
- FROM clause
  - and joins 115
  - SELECT statement 1142



PATINDEX function 943  
 PI function 939  
 PLAN function 967  
 POWER function 939  
 proc\_role SQL Server function 573  
 property function 954  
 property\_description function 954  
 property\_name function 954  
 property\_number function 954  
 QUARTER function 949  
 RADIANS function 939  
 RAND function 939  
 REMAINDER function 939  
 REPEAT function 943  
 reserved\_pgs SQL Server function 573  
 RIGHT function 943  
 ROUND function 939  
 rowcnt SQL Server function 573  
 RTRIM function 943  
 SECOND function 949  
 SECONDS function 949  
 show\_role SQL Server function 573  
 SIGN function 939  
 SIMILAR function 943  
 SIN function 940  
 SOUNDEX function 109, 943  
 SQRT function 940  
 string 566  
 STRING function 943  
 SUBSTR function 944  
 SUM function 937  
 suser\_id SQL Server function 573  
 suser\_name SQL Server function 573  
 system 336, 573  
 TAN function 940  
 text and image 572, 573  
 today 1331  
 TODAY function 949  
 TRACEBACK function 302, 967  
 Transact-SQL 565, 566, 568, 570, 572, 573  
 TRIM function 944  
 TRUNCATE function 940  
 tsequal 552  
 tsequal SQL Server function 573  
 UCASE function 944  
 used\_pgs SQL Server function 573  
 user\_id SQL Server function 573  
 user\_name SQL Server function 573  
 user-defined 1005, 1130

valid\_name SQL Server function 573  
 valid\_user SQL Server function 573  
 WEEKS function 949  
 YEAR function 950  
 YEARS function 950  
 YMD function 950

## G

GET DATA statement  
   syntax 1081  
 GET OPTION statement  
   syntax 1084  
 getdate Transact-SQL function 568  
 getting commands 92  
 global variables  
   about 556, 903  
   in procedures 903  
   selecting 903  
 GOTO statement  
   Transact-SQL statement 618  
 GRANT CONSOLIDATE 466  
 GRANT CONSOLIDATE statement  
   syntax 1089  
 GRANT PUBLISH 434, 442, 464  
 GRANT PUBLISH statement  
   syntax 1091  
 GRANT REMOTE 434, 442, 466  
 GRANT REMOTE statement  
   syntax 1092  
 GRANT statement  
   and concurrency 260  
   CONNECT permissions 394  
   creating groups 386  
   DBA authority 381  
   example 140  
   group membership 387  
   inheriting permissions 383  
   new users 380  
   passwords 381  
   procedures 384  
   resource authority 381  
   syntax 1085  
   table permissions 382  
   Transact-SQL 589  
   UPDATE permissions on views 394  
   views permissions 382  
   WITH GRANT OPTION 383



- without password 389
- granting consolidate permissions 466
- granting permissions
  - about 379
  - on procedures 379
- granting publish permissions 464
- granting remote permissions 466
- GROUP BY ALL clause
  - Transact-SQL 593
- GROUP BY clause 123
  - and views 218
  - SELECT statement 1143
- GROUP permissions
  - about 386
  - not inheritable 386
- grouped data 121
- groups
  - adding 60
  - adding users, using Sybase Central 61
  - creating 60, 386
  - creating in Sybase Central 387
  - granting membership in 387
  - in SQL Server and SQL Anywhere 541
  - managing 386
  - of services 528
  - permissions 379, 388
  - PUBLIC 390
  - special 389
  - SYS 390
  - without passwords 389

## H

- HAVING clause
  - and GROUP BY clause 125
  - SELECT statement 1143
- header files
  - for Embedded SQL 659
- heading
  - ISQL option 1165
- heading name 1142
- Help 86
  - accessing from ISQL 71
- HELP statement
  - syntax 1094
- hexadecimal constants
  - about 921
  - data type 1162

- HintUsed property 955, 959
- HLI
  - about 654
- host variables 897
  - about 667
  - bind variables 692
  - Transact-SQL compatibility 1163
  - types 668
- host\_id SQL Server function 573
- host\_name SQL Server function 573
- hot links 767
- HOUR function 948
- HOURS function 948
- HP UX 809

## I

- I/O
  - estimates 334
  - idle 402
  - operations 817
- IBM AIX 809
- IBM C Set ++ 656
- icon
  - for running services 524
- icons
  - used in manuals 10
- identifiers
  - uniqueness of 547
  - valid 897
- identity column 554
  - Transact-SQL 550
- idle I/O
  - task 402
- IdleCheck property 959
- IdleChkpt property 959
- IdleChkTime property 959
- IdleWrite property 959
- IF expression 906
- IF statement 283
  - and passthrough mode 515
  - syntax 1095
  - Transact-SQL statement 618
- IF UPDATE clause
  - in triggers 616, 982, 1028
- IFNULL function 966
- IMAGE data type
  - Transact-SQL 551

- import libraries
  - alternative to 723
  - for Embedded SQL 660
- importing data 364, 370, 372
  - conversion errors 372
  - INPUT statement 1098
  - performance 373
- IN conditions 110, 578, 912
- INCLUDE statement
  - and the SQLCA 673
  - syntax 1097
- inconsistency 248
- incremental backups
  - about 826
  - defined 411
  - performing 413
- IndAdd property 955, 959
- index\_col SQL Server function 573
- INDEX\_ESTIMATE function 966
- indexes
  - about 1007
  - and foreign keys 1008
  - and primary keys 1008
  - and table use 1008
  - and views 1007
  - automatically created 1008
  - balanced indexes 327
  - clustering 585
  - creating 223, 326, 585, 1007
  - creating, in Sybase Central 224
  - dropping 223, 1053
  - dropping, in Sybase Central 224
  - how indexes work 327
  - in the system tables 1338, 1340
  - inspecting 224
  - naming 1008
  - on primary keys 223
  - owner 1007
  - system views 1358
  - Transact-SQL 585
  - unique 1007
  - unique names 1008
  - uniqueness of names 547
  - using 223
    - using to improve performance 326
    - using with views 223
- indicator variables 897
  - about 671
- IndLookup property 955, 959
- inequality, testing for 106
- initialization files 810
- initialization functions
  - listing 702
- initializing databases 841
- inner joins
  - FROM clause 1077
  - in procedures 292
  - using 149
- input format
  - ISQL option 1165
- INPUT statement
  - about 371
  - syntax 1098
- INSERT
  - multi-row 686, 1062, 1125
  - PUT statement 1124
  - wide 686, 1062, 1125
- insert mode 91
- INSERT permission 382
- INSERT statement
  - about 371
  - and data import 372
  - and integrity 227
  - examples 128, 133
  - FROM SELECT clause 372
  - syntax 1102
  - Transact-SQL 590
  - truncation of strings 1162
- INSERTSTR function 942
- INT data type 921
  - Transact-SQL 549
- INTEGER data type 921
  - Transact-SQL 549
- integer data types 157
  - Transact-SQL 549
- integer overflow
  - ANSI behavior 1158
- integrity
  - about 225, 226
  - constraints 227, 229, 1022
  - reviewing 244
  - rules in the system tables 244
  - SQL statements for 229
- interface library
  - dynamic loading 723
- Internet
  - and SQL Remote 458
  - e-mail 458

- interrupt 96
- interrupts
  - handling 719
- INTO clause 291, 682
  - SELECT statement 1142
- invalid data 227
  - types of 226
- IS FALSE conditions 914
- IS NULL conditions 578, 913
- IS TRUE conditions 914
- IS UNKNOWN conditions 914
- ISNULL function 966
- isolation levels 250
  - about 251
  - and locks 251
  - and serializability 257
  - changing within a transaction 252
  - choosing 256
  - cursors 1113
- ISOLATION\_LEVEL option 1150
- ISQL
  - about 40
  - and Open Server Gateway 644
  - character mode 81
  - command line 846
  - Command window 72
  - connecting to a database 1000
  - displaying a list of tables 112
  - erasing databases 210
  - error handling 1167
  - executing commands 72, 92
  - for DOS 81
  - for Microsoft Windows 67
  - for Microsoft Windows NT 67
  - for OS/2 67
  - for QNX 81
  - getting commands 72, 92
  - interrupting commands 79
  - loading commands 72, 92
  - options 1165
  - saving commands 72, 92
  - starting from Sybase Central 848
  - statistics window 322
  - using 67, 205
- ISQL command delimiter 309
- ISQL\_LOG
  - ISQL option 1167

## J

- Japanese character set 352
- joining lines 91
- joins
  - about 111
  - and deletes 1045
  - and updates 1183
  - automatic 1311
  - cross product 114
  - execution plans 329
  - FROM clause 1075
  - inner joins 149, 292
  - key 1311
  - natural 1311
  - or subqueries 148
  - outer joins 149, 291, 564
  - SELECT statement 1142
  - Transact-SQL 564, 591
  - updates based on 1184

## K

- key joins
  - FROM clause 1077
  - using 118, 328
- keyboard 84
- keys
  - about 183
  - and performance 322
  - assigning 192
  - choosing 201
  - creating 214
  - foreign 183
  - primary 183
- keywords
  - listing of 1316
  - NON\_KEYWORDS option 1161
  - turning off 1161
- Korean character set 352

## L

- labels
  - for statements 618, 898
- language support 354
  - multibyte character sets 352
- laptop computers 451

- and replication 451
- and SQL Remote 451
- and transactions 262
- large databases
  - index storage 1007
- LAST USER
  - special constant 901
- LastIdle property 955
- LastReqTime property 955
- LCASE function 942
- lct\_admin SQL Server function 573
- leaf page
  - and indexes 327
- leap years 932
- LEAVE statement
  - syntax 1104
- LEFT function 942
- left, moving screen 93
- LegalCopyright property 959
- LegalTrademarks property 959
- length 692
- LENGTH function 942
- libraries
  - for Embedded SQL 660
- library functions
  - and Embedded SQL 701
- LIKE conditions 577, 909
- LIKE operator 108
- line length
  - SQLPP output 893
- line numbers 893
- links 767
- LIST function 122, 937
- literal strings 898, 900
- live backups 415
- LOAD DATABASE statement
  - Transact-SQL 588
- LOAD TABLE statement
  - about 370
  - syntax 1105
- LOAD TRANSACTION statement
  - Transact-SQL 588
- loading commands 92
- loading tables 1105
- local variables
  - about 556
  - and result sets 624
- LocalSystem account 524
- LOCATE function 942
- locks 248, 250
  - exclusive 250
  - how locking works 250
  - nonexclusive 250
  - phantom 250
  - read 250
  - write 250
- LockTablePages property 955, 959
- log files 871
  - about 42
  - checkpoint 401
  - erasing 42
  - rollback 403
  - transaction 403
- LOG function 939
- log translation utility 404, 418
- LOG10 function 939
- LogFreeCommit property 955, 959
- logging on 84
  - to a database 48, 70
- logical operators 579
  - and three-valued logic 914
- LogName property 963
- logon options 524
- LogRewrite property 955, 959
- LogWrite property 955, 959
- LONG BINARY
  - replication 492
- LONG BINARY data type 926
  - Transact-SQL 551
- long column names
  - retrieving 1051
- long commands 105
- LONG VARCHAR
  - replication 492
- LONG VARCHAR data type 918
  - Transact-SQL 550
- lookup
  - and indexes 327
- LOOP statement 283
  - and passthrough mode 515
  - in procedures 297
  - syntax 1108
- lost updates
  - and transactions 256
- LOTUS file format 1165, 1167
- Lotus Notes
  - and SQL Remote 429, 458, 461
- LTM

transaction log options 871  
LTRIM function 942

## M

Macintosh

ODBC programming for 762

MainHeapPages property 959

mandatory foreign keys 241

MAPI

system procedures 1323, 1324, 1325

MAPI link 429

MAPI message type

about 976

MAPI messages

about 1015, 1058

MapPages property 959

MASTER service 643

mathematical expressions 899

MAX function 122, 937

maximum function 122, 937

MaxIO property 959

MaxRead property 959

MaxWrite property 959

media

recovery from failure of 416

media failure 403

recovery from 406

membership

of groups 387

memory 718

allocation

DOS 718

QNX 718

and caching 320

usage 719

menu 85

Message Agent 424, 447

about 496

and message tracking 500, 502

and recovery 498

and security 497

and transaction log management 503, 504

command line 863

delivering messages 500, 502

message links

parameters 459

MESSAGE statement

in procedures 301

syntax 1109

message-based replication 423

messages

and synchronizing databases 487

creating 586

delivering 500, 502

in SQL Remote 500, 502

receiving 447

sending 447

tracking 500, 502

types 976, 1015

dropping 1058

Microsoft C

support for 656

Microsoft Visual C++

support for 656

millenium issues 930

MIN function 122, 937

minimum function 122, 937

MINUTE function 948

MINUTES function 948

mirror

transaction log 403, 406, 408

mobile workforces 428, 451, 472

MOD function 939

MONEY data type

Transact-SQL 551

money data types

Transact-SQL 551

monitoring

services 531

monitoring performance 338

MONTH function 948

MONTHNAME function 948

MONTHS function 948

moving screen 93

multibyte character sets 352

multiple result sets from procedures 293

multiple row queries 295

and cursors 678

multi-row fetches 686, 1070

OPEN statement 1113

multi-row inserts 686, 1062

multi-row puts 686, 1125

multi-tier installations

passthrough statements 514

## N

- Name property 955, 959, 963
- national language support 354
  - multibyte character sets 352
- natural joins
  - FROM clause 1076
- NCHAR data type
  - and OmniCONNECT 650
  - Transact-SQL 550
- NEAREST\_CENTURY option 1160
- NetWare
  - about 817
  - and SQL Remote 457, 460
  - cache buffers 817
  - file system 817
  - ISQL 81
- network server
  - connecting to 26
  - same-machine connections 26
- new databases 525
- next\_connection function 954
- next\_database function 954
- NLMs
  - calling from functions 314
  - calling from procedures 314
- NO SCROLL cursors 1039
- NodeAddress property 955
- NON\_KEYWORDS database option 1161
- nonexclusive locks 250
- non-repeatable reads 248, 251
- normal forms 191
- NOT conditions 913
- not found warning 295, 678
- NOT NULL constraint 228
- Notes
  - and SQL Remote 458
- NOW function 949
- NT performance monitor 338
- NT services
  - example code 746, 760
  - startup options 522
- NULL value 691, 965
  - about 1110
  - allowed in columns 128
  - and columns 200
  - and indicator variables 671
  - and output 369
  - ANSI behavior 1159

- column default 546
- default 233
- integrity action 242
- NULLS option 1167
  - Transact-SQL behavior 1159
- NULLS option
  - ISQL option 1167
- NUMBER function
  - about 966
  - and updates 1184
  - limitations 966
- number of rows 1347
- Number property 955
- numbers 900
  - defaults 233
- NUMERIC data type 921
  - Transact-SQL 550
- numeric functions 565, 938
- numeric precision
  - database option 1155
- NVARCHAR data type
  - and OmniCONNECT 650
  - Transact-SQL 550

## O

- object\_id SQL Server function 573
- object\_name SQL Server function 573
- objects
  - name prefixes 391
  - qualified names 391
- ODBC
  - about 26
  - Administrator 170, 173, 174, 179
  - and character sets 349
  - conformance 169
  - connecting from 169
  - cursors 756
  - data sources 163, 173
    - about 162
  - data sources, adding 174
  - data sources, modifying 179
  - DOS and 169, 180
  - driver manager 170
  - interface 34
  - language dll 172
  - Macintosh applications 762

- ODBC.DLL 170
- ODBC.INI 170, 173, 174, 179
- ODBCINST.INI 170, 173
- OS/2 and 169, 179
- programming 752
- QNX and 169, 180
- SQL Anywhere driver 170
- static cursors 1039
- support for 653
- unsupported functions 759
- Windows and 170
- Windows NT and 170
- ODBC programming 751
- offline backups
  - defined 411
- OmniCONNECT
  - about 630, 649
  - and Open Server Gateway 630
  - data types 650
  - support for 649
  - support limitations 650
- ON EXCEPTION RESUME 300
- ON\_ERROR
  - ISQL option 1167
- online backups 711
  - about 826
  - defined 411
- online Help
  - contents 65
  - documentation 9
  - for Sybase Central 65
  - full-text search 66
  - index 66
  - searching 66
- Open Client data types
  - description limitations 640
  - mapping from SQL Anywhere data types 637
  - mapping to SQL Anywhere data types 636
  - problem free 634
  - unsupported in SQL Anywhere 641
  - value range limitations 638
- Open Server Gateway 855, 856
  - adding 642
  - and database options 644
  - and OmniCONNECT 630
  - and Replication Server 630
  - and two-phase commit 648
  - configuring 642
  - connecting to 644
  - data type mappings 634
  - database requirements 633
  - information 855
  - limitations 648
  - master service for 643
  - performance 644
  - starting 643
  - stopping 856
  - verbose mode 644
- Open Servers 853, 855, 856
  - connecting to 644
  - master service 643
  - query service 643
  - starting 643
- OPEN statement 733
  - about 678
  - in procedures 295
  - QUERY\_PLAN\_ON\_OPEN option 1161
  - syntax 1112
- opening cursors 1161
- operating systems
  - supported 5
- operators
  - comparison operators 908
  - precedence of 564
  - Transact-SQL 563, 577
- optimizer
  - about 333
  - cost based 333
  - estimates 333
- optional foreign keys 241
- options
  - about 1150
  - ALLOW\_NULLS\_BY\_DEFAULT 546, 1157, 1158
  - ANSI\_BLANKS 1157, 1158
  - ANSI\_INTEGER\_OVERFLOW 1158
  - ANSI\_PERMISSIONS 1159
  - ANSINULL 1159
  - AUTOMATIC\_TIMESTAMP 546, 1157, 1159
  - BACKGROUND\_PRIORITY 1150, 1151
  - BLOCKING 1150
  - CHAINED 1160
  - CHECKPOINT\_TIME 1150
  - CLOSE\_ON\_ENDTRANS 1160
  - CONVERSION\_ERROR 1157
  - CONVERSION\_ERROR option 1160
  - COOPERATIVE\_COMMIT\_TIMEOUT 1150

- COOPERATIVE\_COMMITS 1150
- DATE\_FORMAT 1150
- DATE\_ORDER 1150
- DELAYED\_COMMIT\_TIMEOUT 1150
- DIVIDE\_BY\_ZERO\_ERROR 1160
- FIRE\_TRIGGERS 1160
- GET OPTION statement 1084
- in the system tables 1341
- ISOLATION\_LEVEL 1150
- NEAREST\_CENTURY option 1160
- NON\_KEYWORDS option 1161
- PRECISION 1150
- QUERY\_PLAN\_ON\_OPEN 1157, 1161
- QUOTED\_IDENTIFIER 546, 562, 1157, 1161
- RECOVERY\_TIME 1150
- retrieving 1084
- RI\_TRIGGER\_TIME 1162
- ROW\_COUNTS 1150
- SCALE 1150
- set by Open Server Gateway 644
- setting 1149
- setting ISQL options 998
- SQL\_FLAGGER\_ERROR\_LEVEL 1162
- SQL\_FLAGGER\_WARNING\_LEVEL 1162
- system views 1358, 1361
- THREAD\_COUNT 1150
- TIME\_FORMAT 1150
- Transact-SQL 594
- TSQL\_HEX\_CONSTANT 1162
- TSQL\_VARIABLES 1163
- WAIT\_FOR\_COMMIT 1150
- OR keyword 110, 913
- ORDER BY clause 105, 1143
  - and performance 331
  - examples 103
- OS/2
  - and ISQL 67
  - and SQL Remote 457
  - DOS client applications 32
  - Windows 3.x client applications 32
- OS/2 DLL
  - for Embedded SQL 701
- outer joins
  - and subqueries 905
  - FROM clause 1077, 1078
  - in procedures 291
  - operators 564
  - Transact-SQL 564, 591

- using 149
- outer references
  - defined 148
- output length
  - ISQL option 1169
- output redirection 368
- OUTPUT statement
  - about 368
  - syntax 1115
- owners
  - about 377
  - assigning ownership 389

## P

- page size
  - and performance 320
  - setting 841
- PageRelocations property 959
- pages
  - displaying usage in database files 837
- PageSize property 963
- parameters
  - command files 1126
  - to command files 154
  - to functions 122
- PARAMETERS statement
  - syntax 1118
- PASSTHROUGH 513
- passthrough mode 513
  - starting 1119
  - stopping 1119
  - uses 514
- PASSTHROUGH statement
  - syntax 1119
- passthrough statements
  - and multi-tier installations 514
- password 48, 84
- passwords
  - and connecting 162
  - and permissions 140, 394
  - changing 381, 1086
  - defined 163
  - in the system tables 1352
  - using with ISQL 70
- PATINDEX function 943
- pattern matching 108, 577, 909, 943
  - and case-sensitivity 910



- and collations 910
  - limits 910
  - maximum length of pattern 910
- pausing
  - services 526
- PendingReq property 959
- performance
  - and cache size 320, 817
  - and foreign keys 325
  - and indexes 223
  - and multiple table queries 328
  - and page size 320, 327
  - and primary keys 324
  - disk fragmentation 209
  - improving 126, 319
  - monitoring 319, 336, 337, 338
  - of bulk operations 373
  - setting priorities 1151
  - temporary tables 332
  - transaction log 403
  - using ISQL to examine 322
- performance monitor
  - NT 338
  - starting 338
- permissions 158
  - about 376
  - assessing 396
  - changing 381
  - CONNECT authority 1086
  - consolidate 466, 1089
  - creating groups 386
  - DBA authority 376, 1086
  - execute 384, 1088
  - for triggers 377, 385
  - GRANT statement 1085
  - granting 378, 381, 382
  - granting connect permissions 380
  - granting group 379
  - granting group membership 387
  - granting individual 380
  - granting on procedures 384
  - granting passwords 380, 381
  - granting resource permissions 377, 381
  - group 379
  - GROUP authority 1086
  - in Sybase Central 383, 384
  - in the system tables 1333, 1348
  - individual 380
  - inheriting 383, 386
  - inspecting 397
  - managing 375
  - MEMBERSHIP 1086
  - of groups 388
  - on procedures 395
  - on tables 221, 378, 382
  - on triggers 279
  - on views 140, 221, 382
  - publish 434, 442, 464
  - remote 434, 442, 466
  - resource authority 377, 1086
  - revoking 385, 1132
  - revoking CONSOLIDATE 1134
  - revoking PUBLISH 1135
  - setting 58
  - SQL Server compatibility 541, 589
  - SYSCOLAUTH system view 1356
  - system views 1360
  - the right to grant 383
  - UPDATE, on views 394
  - users 375
  - WITH GRANT OPTION 383
- phantom
  - locks 250
  - reads 251
  - rows 248
- Pharlap
  - support for 660
- PI function 939
- place holders
  - and dynamic SQL 683
- PLAN function 967
- Platform property 959
- platforms
  - supported operating systems 5
- Polling frequency
  - for services 519
- Port property 955
- portable computers
  - and databases 262
  - and decision support applications 263
  - and multiuser databases 262
  - and SQL Anywhere 262
  - runtime applications 263
- power failure
  - recovery from 416
- POWER function 939
- PowerBuilder
  - database connections 166

- preallocating space for the transaction log 972
- precedence of operators 564
- PRECISION option 1150
- predicates
  - about 907
- PREPARE statement 683
  - syntax 1120
- PREPARE TO COMMIT statement
  - syntax 1123
  - two-phase commit 260
- PREPARE TRANSACTION statement
  - and Open Server Gateway 648
- prepared statement 1120
- prepared statements 1062
  - in the system tables 1348
- PrepStmt property 955
- previous commands 94
- primary key errors 476
  - in SQL Remote 476
- primary keys 183
  - about 966
  - adding 53
  - and concurrency 259
  - and entity integrity 239
  - and indexes 223
  - and integrity 1310
  - and performance 324
  - and replication 478, 479
  - and SQL Remote 478, 479
  - and transaction log 404
  - choosing 201
  - creating 53, 214
  - entity integrity 240
  - generation 259
  - identifying rows by 117
  - in the system tables 1334, 1337, 1347
  - integrity constraints 1023
  - using to improve performance 322
- primary table
  - in the system tables 1337
- PRINT statement 619
- proc\_role SQL Server function 573
- ProcedurePages property 959
- procedures 1010, 1051, 1121
  - about 266
  - allowed statements in 286
  - altering 974
  - and control statements 283
  - and data definition statements 286, 1064
  - and dynamic SQL statements 1064
  - and efficiency 267
  - and passthrough mode 515
  - and security 267, 393, 395
  - and SQL Remote 490, 491
  - and standardization 267
  - and Transact-SQL compatibility 608
  - and Transact-SQL support 606
  - benefits of 267
  - calling 269, 288, 617
  - catalog 601, 602
  - command delimiter and 309
  - control statements 614
  - creating 268, 1009
  - cursors in 295
  - default values for parameters 287, 288
  - dropping 270, 1053
  - dynamic statements in 307
  - Embedded SQL 696
  - error handling in Transact-SQL 620
  - error handling 300, 304
  - errors in 300
  - EXECUTE IMMEDIATE 286
  - EXECUTE IMMEDIATE statement 307
  - executing 269, 288, 617
  - external 314
  - FOR statement 299
  - in ODBC 757
  - invoking 269
  - multiple result sets from 293
  - OUT parameters 270
  - owner 377
  - parameters 287, 288
  - permissions for creating 377
  - permissions on 379, 384
  - RAISERROR statement 620
  - replicating 490, 491, 974
  - result sets from 292
  - returning results from 270, 290
  - returning values from 1130
  - setting permissions 58
  - SQL statements allowed in 286
  - system 601
  - table names in 309
  - testing 271, 309
  - Transact-SQL compatibility 615
  - Transact-SQL CREATE PROCEDURE statement 615
  - Transact-SQL overview 610

- Transact-SQL RETURN statement 621
- translation of 58, 608
- translation of, using Sybase Central 608
- using 265, 268
- using cursors in 297
- using ON EXCEPTION RESUME 300
- variable result sets from 294
- viewing 57
- viewing Transact-SQL syntax 58
- warnings in 300
- writing 309
- ProcessTime property 955
- product name
  - retrieving 1327
- ProductName property 959
- ProductVersion property 959
- program structure
  - Embedded SQL 662
- programming interfaces
  - about 25, 34
  - DDE 35
  - embedded SQL 34
  - HLI 35
  - ODBC 34
  - supported by SQL Anywhere 34
- projections
  - SELECT statement 1142
- properties
  - connection 955, 1321
  - database 963, 1321
  - engine 959, 1321
- property function 954
- property\_description function 954
- property\_name function 954
- property\_number function 954
- PUBLIC group 390
  - in the system tables 1352
- publications 424
  - and primary keys 478, 479
  - and referential integrity 478
  - and subqueries 480, 482
  - and updates 1185
  - creating 436, 444, 1013
  - designing 475, 476, 477, 478, 479, 1185
  - dropping 473, 1057
  - example 450
  - for many subscribers 472
  - notes on 474
  - selected columns 471

- setting up 470
  - using a WHERE clause 472
  - using SUBSCRIBE BY clause 472
  - whole tables 470
- publish permissions 434, 442, 1091
  - granting 464
  - managing 463
  - remote permissions 434, 442
  - revoking 464, 1135
- publisher 435, 443, 464, 900
  - address 976, 1015, 1058
  - GRANT PUBLISH statement 1091
  - remote 1092
- PURGE clause
  - FETCH statement 1070
- PUT
  - multi-row 686
  - wide 686
- PUT statement
  - and integrity 227
  - multi-row 1125
  - syntax 1124
  - wide 1125

## Q

- QNX 809
  - and SQL Remote 457
  - ISQL 81
  - ISQL Statistics window 323
  - memory
    - allocation 718
    - SQL Anywhere Client 29, 37
- qualified object names 391
- QUARTER function 949
- queries
  - joins 591
  - optimization 333
  - SELECT statement 1141
  - sorting results of 331
  - Transact-SQL 592
- QUERY service 643
- QUERY\_PLAN\_ON\_OPEN option 1157, 1161
- quotation marks
  - and identifiers 897
  - in SQL 215, 394
  - using 105

QUOTED\_IDENTIFIER option 546, 562, 1157

## R

- RADIANS function 939
- RAISERROR statement 620
- RAND function 939
- range 106
- Rational DOS4G
  - support for 660
- read locks 250
- READ statement
  - syntax 1126
- ReadHint property 955, 959
- READTEXT statement
  - Transact-SQL 591
- REAL data type 921
  - Transact-SQL 550
- REBUILD 42, 857
- rebuilding databases 857
- recalling commands 94
- receiving messages 447
- recover
  - rapid 415
- recovery
  - about 399
  - and SQL Remote 498
  - from media failure 406
  - time 1155
  - transaction log 403
  - uncommitted changes 418
- RECOVERY\_TIME option 1150
- redirection 368
- REFERENCE permissions 382
- referential integrity 1310
  - about 228
  - and replication 478
  - and SQL Remote 478
  - breached by client application 242
  - checking 243
  - enforcing 239, 240
  - FROM clause 1074
  - losing 241
- referential integrity actions
  - and triggers 1162
- registry
  - and SQL Remote 459
- registry entries 810
- relational databases
  - about 182
  - concepts 182
  - terminology 182
- relations
  - and entities 182
- relationships
  - between entities 185
  - in the system tables 1337
  - resolving 195
- RELEASE SAVEPOINT statement 258
  - syntax 1127
- RelocatableHeapPages property 959
- REMAINDER function 939
- remote databases 422
  - and remote permissions 467
  - setting up 438, 445, 446
- remote permissions 1092
  - granting 466
  - managing 463
  - revoking 466, 468, 1136
- remote users
  - REVOKE REMOTE statement 1136
- removing
  - services 521
- renaming
  - columns 980
  - tables 980
- REPEAT function 943
- repeatable reads 248
- REPLICATE\_ALL
  - replication option 1163
- replicating procedures 490, 491
- replicating triggers 490, 491
- replication 859
  - administering 427
  - and mobile workforces 451
  - and triggers 479
  - backup procedures 503, 504
  - by e-mail 423
  - case studies 451, 453
  - conflicts 475, 507
  - data types 492
  - dbcc 871
  - DBREMOTE 863
  - design 475
  - errors 507
  - Message Agent 863
  - message-based 423

**xxx**

- of blobs 492
- of control statements 515
- of cursor statements 515
- of procedures 490, 491, 974
- of SQL statements 513
- of stored procedures 515
- of trigger actions 1160
- of triggers 490, 491
- options 1163
- passthrough mode 513
- primary key errors 476, 478, 479
- publications 424
- referential integrity errors 478
- replication server 871
- server-to-laptop replication 451
- server-to-server 453
- setup examples 451
- subscriptions 424
- synchronization 859
- transaction log and 427
- transaction log management 503, 504
- UPDATE conflicts 477
- upgrading databases 505
- replication options
  - REPLICATION\_ERROR 508, 1163
  - VERIFY\_THRESHOLD 1164
- Replication Server
  - and Open Server Gateway 630
  - and SQL Remote 498
  - Open Servers 853, 855, 856
- REPLICATION\_ERROR
  - replication option 508, 1163
- Req property 959
- ReqType property 955
- request processing
  - and Windows 3.x 716
- requests
  - aborting 715
- reserved words *See* keywords
- reserved\_pgs SQL Server function 573
- resident program 814
- RESIGNAL statement
  - in procedures 305
  - syntax 1128
- RESOLVE UPDATE triggers 507, 509, 510
- resource authority 381, 386
  - about 377
  - in the system tables 1352
  - not inheritable 386
- resource management 589
- restore *See* recovery
- restoring
  - databases 399
- RESTRICT 1025
- restrict action 242
- restrictions *See* WHERE clause
- RESULT clause
  - about 1009
- result sets
  - and local variables 624
  - and temporary tables 624
  - from procedures 271, 292
  - multiple 293
  - shape of 294, 1010, 1051, 1121
  - variable 294, 1010, 1051, 1121
- RESUME statement
  - syntax 1129
- return codes 808, 813
- RETURN statement
  - syntax 1130
  - Transact-SQL 621
- returning results from procedures 290
- REVOKE CONSOLIDATE 466
- REVOKE CONSOLIDATE statement
  - syntax 1134
- REVOKE PUBLISH 464
- REVOKE PUBLISH statement
  - syntax 1135
- REVOKE REMOTE 466, 468
- REVOKE REMOTE statement
  - syntax 1136
- REVOKE statement
  - about 385
  - and concurrency 260
  - syntax 1132
  - Transact-SQL 589
- revoking consolidate permissions 466
- revoking publish permissions 464
- revoking remote permissions 466, 468
- REXX 802
- RI\_TRIGGER\_TIME option 1162
- RIGHT function 943
- right, moving screen 93
- Rlbk property 955
- role name 898
- role names
  - about 1024
- roles

- compatibility with SQL Server 540
- roll forward 403
- ROLLBACK
  - TO SAVEPOINT statement 258
- rollback log 403
  - and recovery 403
- ROLLBACK statement 258
  - about 130
  - and procedures 308
  - and transactions 246
  - examples 132
  - in compound statements 285
  - ROLLBACK TRIGGER 1139
  - syntax 1137
  - Transact-SQL 592
- ROLLBACK TO SAVEPOINT statement
  - syntax 1138
- ROLLBACK TRIGGER statement
  - syntax 1139
- RollbackLogPages property 955, 959
- root file
  - about 28
  - defined 31
- ROUND function 939
- rounding
  - SCALE option 1155
- row counts 1155
- row not found 295, 678
- ROW\_COUNTS option 1150
- rowcnt SQL Server function 573
- rows
  - about 101, 182
  - selecting 105
- RTDSK50 37
- RTDSK50S 37
- RTDSK50W 37
- RTRIM function 943
- RTSQL
  - command line 846
- rules
  - Transact-SQL 585
- runtime database engine 37
- runtime system 37

## S

- sa\_conn\_info system procedure 1321
- sa\_conn\_properties system procedure 1321

- sa\_db\_info system procedure 1321
- sa\_db\_properties system procedure 1321
- sa\_eng\_properties system procedure 1321
- sample database
  - connecting to 70
- SAVEPOINT statement
  - and transactions 258
  - syntax 1140
- savepoints 308
  - about 898
  - and procedures 308
  - and triggers 308
- RELEASE SAVEPOINT statement 1127
- ROLLBACK TO SAVEPOINT statement
  - 1138
  - within transactions 258
- saving commands 92
- saving statements 1167
- SCALE option 1150
- schema
  - and system tables 1329
  - creating 587, 1017
  - Transact-SQL 587
  - viewing 49
- screen
  - moving left or right 75
- SCROLL cursors 1039
- sdefaultx
  - user ID 232
- search conditions
  - about 907
  - ALL conditions 578, 912
  - and GROUP BY clause 125
  - AND keyword pattern matching 108
  - and logical operators 579
  - and three-valued logic 914
  - ANY conditions 578, 912
  - BETWEEN conditions 577, 909
  - comparison conditions 908
  - date comparisons 106
  - estimates 579
  - EXISTS conditions 578, 912
  - IN conditions 578, 912
  - introduction to 105
  - IS NULL conditions 578, 913
  - LIKE conditions 577, 909
  - NOT conditions 913
  - short cuts for 110
  - subqueries 141, 908

- Transact-SQL 576, 577, 578, 579
  - truth value conditions 914
- SECOND function 949
- SECONDS function 949
- security 84
  - about 375
  - and procedures 267
  - passwords 70
  - services 524
  - tailored 393
  - using views for 393
- SELECT 89
- SELECT LIST 685
  - DESCRIBE statement 1049
  - SELECT statement 1142
- SELECT permissions 382
- SELECT statement
  - about 99, 141
  - access plans 328
  - and ISQL 74
  - and performance 326
  - examples 1355
  - FROM clause 1074
  - INTO clause 291
  - keys and query access 322
  - syntax 1141
  - to view variable values 560
  - Transact-SQL 592
- SEND AT clause 1089, 1092
  - publish 1091
- SEND EVERY clause 1089, 1092
- sending messages 447
- sequential searching 324
- serializable transactions 257
- server
  - defined 31
  - name 31
- ServerName connection parameter 163
- server-to-server replication 453
- services
  - about 518, 520, 525, 526
  - account 524
  - adding 520
  - adding new databases 525
  - command-line switches 523
  - configuring 522
  - deleting 521
  - dependencies 528, 529
  - example code 746, 760
  - executable file 524
  - failure to start 523
  - groups 528
  - icon on the desktop 524
  - load ordering groups 528
  - managing 519
  - monitoring 531
  - more than one server 528, 530
  - multiple 528, 530
  - options 523
  - parameters 522
  - pausing 526
  - removing 520, 521
  - security 524
  - service manager 532
  - setting login options 524
  - setting logon options 524
  - starting 526
  - starting order 529
  - startup options 522
  - stopping 526
  - Windows NT Control Panel 532
- SET CONNECTION statement
  - syntax 1147
- SET DEFAULT 242, 1025
- SET NULL action 242, 1025
- SET OPTION statement
  - syntax 1149
  - Transact-SQL 546, 594
- SET SQLCA statement
  - syntax 1170
- SET statement
  - in procedures 290
  - syntax 1145
  - Transact-SQL 594
- setting up a remote database 438, 445
- setting up publications 470
- setting up subscriptions 483
- setting up the consolidated database 433, 442
- Shift-JIS collation 352
- show\_role SQL Server function 573
- SIGN function 939
- SIGNAL statement
  - in procedures 301
  - syntax 1171
- SIMILAR function 943
- SIN function 940
- single row queries 290, 291
  - in Embedded SQL 677

- SMALLDATETIME data type
  - Transact-SQL 552
- SMALLINT data type 921
  - Transact-SQL 549
- SMALLMONEY data type
  - Transact-SQL 551
- SMTP
  - and SQL Remote 458
  - e-mail 458
- SMTP link 429
- software
  - components 808
  - DBBACKUP 826
  - DBCOLLAT 830
  - DBERASE 835
  - DBEXPAND 874
  - DBINFO 838
  - DBINIT 841
  - DBLOG 871
  - DBOS50 853
  - DBOSINFO 855
  - DBOSSTOP 856
  - DBREMOTE 863
  - DBSHRINK 833
  - DBSTOP 867
  - DBTRAN 850
  - DBUNLOAD 878
  - DBUPGRAD 883
  - DBVALID 886
  - DBWRITE 890
  - DBXTRACT 859
  - ISQL 846
  - REBUILD 857
  - return codes 813
  - RTSQL 846
  - SQLPP 892
- SOME conditions 912
- sort position
  - description 359
- sorting
  - in the system tables 1332
  - query results 103, 331
- SOUNDEX function 109, 943
- sp\_addgroup system procedure 601
- sp\_addlogin system procedure 601
- sp\_addmessage system procedure 586, 601
- sp\_addtype system procedure 601
- sp\_adduser system procedure 601
- sp\_changegroup system procedure 601
- sp\_column\_privileges
  - catalog procedure 602
- sp\_columns catalog procedure 602
- sp\_dboption system procedure 601
- sp\_dropgroup system procedure 601
- sp\_droplogin system procedure 601
- sp\_dropmessage system procedure 601
- sp\_droptype system procedure 601
- sp\_dropuser system procedure 601
- sp\_fkeys catalog procedure 602
- sp\_getmessage system procedure 601
- sp\_helptext system procedure 601
- sp\_password system procedure 601
- sp\_pkeys catalog procedure 602
- sp\_special\_columns catalog procedure 602
- sp\_sproc\_columns catalog procedure 602
- sp\_stored\_procedures catalog procedure 602
- sp\_tables catalog procedure 602
- special tables 1329
- speed
  - and performance 319
- SQL 895
  - communicating with 26
  - dialects 606
  - SQL/92 syntax 1162
  - SQL/92 syntax 893
  - values 215
- SQL Anywhere
  - multiplatform support 30
  - new features 15
  - operating systems 30
  - standalone database engine 37
  - statistics 338
- SQL Anywhere Client 37
  - about 29
  - QNX 29, 37
- SQL Anywhere Desktop Runtime System 37, 263
- SQL Anywhere Open Server 853
  - query service for 643
- SQL Anywhere server 518
  - for Windows NT 518
- SQL Preprocessor
  - command-line switches 892
- SQL Remote 419
  - about 455
  - administering 427, 455, 513, 514
  - ALTER REMOTE MESSAGE TYPE
    - statement 976
  - altering publications 975



- and database recovery 498
- and mobile workforces 428, 451
- and procedures 490, 491
- and triggers 490, 491
- articles 1331, 1332
- avoiding conflicts 475
- avoiding primary key errors 476
- avoiding referential integrity errors 478
- backup procedures 503, 504
- case studies 451, 453
- concepts 422
- conflict resolution 507, 510
- consolidate permissions 1089, 1134
- consolidated databases 422
- CREATE REMOTE MESSAGE TYPE
  - statement 1015
- creating publications 436, 444, 1013
- creating subscriptions 483, 1019
- dbremote 447
- DBXTRACT utilities 859
- deployment 494
- designing publications 475
- designing triggers 479
- DROP REMOTE MESSAGE TYPE statement
  - 1058
- dropping publications 473, 1057
- dropping subscriptions 1061
- error reporting 507
- extracting a database 859
- for DOS 457
- for many subscribers 472
- for NetWare 457
- for OS/2 457
- for QNX 457
- for Windows 3.x 457
- for Windows 95 457
- for Windows NT 457
- GRANT PUBLISH statement 1091
- granting publish permissions 434, 442
- granting remote permissions 434, 442
- managing 494
- Message Agent 424, 447
- message delivery 500, 502
- message systems 429
- message tracking 500, 502
- multi-tier installations 514
- options 1163
- passthrough mode 513, 1119
- primary key errors 478, 479
  - publications 424
  - publish permissions 1135
  - publishing a subset of rows 472
  - publishing selected columns 471
  - publishing whole tables 470
  - remote databases 422
  - remote permissions 1092, 1136
  - server-to-laptop replication 451
  - server-to-server replication 453
  - setting up 430, 439
  - setting up a consolidated database 433, 442
  - setting up a remote database 438, 445
  - setup examples 451
  - starting subscriptions 1174
  - stopping subscriptions 1177
  - subqueries 480, 482
  - subscribers 428
  - subscriptions 424
  - synchronization 859
  - synchronizing databases 484, 487
  - synchronizing subscriptions 1178
  - system tables 1331, 1332
  - transaction log management 503, 504
  - unloading databases 505
  - UPDATE conflicts 477
  - upgrading databases 505
  - upgrading installations 494
- SQL variables
  - creating 1031
  - dropping 1060
  - SET VARIABLE statement 1145
- SQL/92
  - conformance 893, 1162
- sql\_needs\_quotes 714
- SQLANY environment variable 810
- SQLANY.INI
  - and SQL Remote 459
- SQLCA 662
  - about 673
  - and threads 699
  - changing 699
  - multiple 700
  - SET SQLCA statement 1170
- sqlcabc 674
- sqlcaid 674
- SQLCancel
  - support for 759
- sqlcode 674
- special constant 900

- values 1191
- SQLCONNECT environment variable 810
- SQLDA
  - about 683, 690
  - allocating 709
  - and ANSI\_BLANKS 1158
  - filling 709
  - freeing 710
  - length field 692
  - strings 710
- sqlda\_storage 710
- sqlda\_string\_length 710
- sqldef.h 747
- SQLEDIT utility 642
- sqlerrd 674
- sqlerrmc 674
- sqlerrml 674
- sqlerror\_message 714
- sqlerrp 674
- sqlen 692
- sqlname 691
- SQLPATH environment variable 811
- SQLPP
  - about 656
  - command line 892
  - running 659
- SQLREMOTE environment variable 459, 811
- SQLSetConnectOption
  - support for 759
- SQLSetStmtOption
  - support for 759
- SQLSTART environment variable 811
- sqlstate 675
  - special constant 900
  - values 1201
- sqlwarn 675
- SQRT function 940
- Start connection parameter 162, 163
- START DATABASE statement
  - syntax 1172
- START ENGINE statement
  - syntax 1173
- START SUBSCRIPTION statement
  - syntax 1174
- starting
  - database engines 1173
  - databases 1172
  - ISQL, from Sybase Central 848
  - services 523, 529
  - Sybase Central 47
  - starting subscriptions 1174
  - startup options
    - for services 522
  - statement labels 618, 898
  - statement-level triggers 611
  - statements *See also* individual statement names
    - control 283
    - SQL, in procedures and triggers 286
    - using compound 283
  - static cursors
    - declaring 1039
  - static SQL 682
  - static SQL authorization 663
  - statistics
    - available 338
    - displaying 338
    - ISQL option 1169
    - monitoring 336
    - monitoring in Sybase Central 337
  - Statistics window 143
  - STOP DATABASE statement
    - syntax 1175
  - STOP ENGINE statement
    - syntax 1176
  - STOP SUBSCRIPTION statement
    - syntax 1177
  - stopping databases 867, 1175
  - stopping subscriptions 1177
  - store-and-forward 423
  - stored procedures
    - and passthrough mode 515
    - creating 1009
  - STRING function 943
  - string truncation
    - database option 1162
  - string type 710
  - strings
    - and host variables 1158
    - concatenating 563, 905
    - concatenation operators 563, 905
    - constants 898, 900
    - defaults 233
    - delimiter 562
    - functions 566
    - literal strings 898
    - Transact-SQL 562
  - structure packing
    - about 657

- subqueries
  - ALL conditions 578
  - and comparisons 145
  - and publications 480, 482
  - ANY conditions 578
  - correlated subqueries 148
  - EXISTS conditions 578
  - in expressions 905
  - in search conditions 908
  - in SQL Remote 480, 482
  - or joins 148
  - using 141
- SUBSCRIBE BY clause 472, 479, 1013
  - subqueries 480, 482
- subscriptions 424
  - and updates 1185
  - creating 436, 444, 483, 1019
  - dropping 1061
  - setting up 483
  - starting 1174
  - stopping 1177
  - synchronizing 484, 487, 1178
- SUBSTR function 944
- subtransactions
  - and precedures 308
  - and savepoints 258
- SUM function 122, 937
- Sun Solaris 809
- suser\_id SQL Server function 573
- suser\_name SQL Server function 573
- Sybase Central
  - about 39
  - adding columns 211
  - adding users to groups 387
  - altering columns 213
  - altering tables 213
  - and column constraints 237
  - and column defaults 231
  - and database objects 204
  - and permissions 383, 384
  - connecting to a database 48
  - creating groups 387
  - creating indexes 224
  - creating tables 211
  - creating users 380
  - creating views 218
  - documentation 65
  - drag and drop columns 54
  - dropping indexes 224
  - dropping views 222
  - erasing databases 210
  - main window 47
  - managing services 518, 519
  - starting 47
  - Starting ISQL from 848
  - Transact-SQL-compatible databases 544
  - translating procedures 608
- synchronization utility 445
- SYNCHRONIZE SUBSCRIPTION statement
  - syntax 1178
- synchronizing databases 445, 446, 484
  - using DBXTRACT 484
  - using the extraction utility 484
  - using the message system 487
- synchronizing subscriptions 1178
- SyncWriteChkpt property 955, 959
- SyncWriteExtend property 955, 959
- SyncWriteFreeCurr property 955, 959
- SyncWriteFreePush property 955, 959
- SyncWriteLog property 955, 959
- SyncWriteRlbk property 955, 959
- SyncWriteUnkn property 955, 959
- syntax rules 895
- SYS group 390
  - in the system tables 1352
- SYSCOLUMNS system view
  - conflicting name 545
- SYSINDEXES system view
  - conflicting name 545
- SYSREMOTEUSER 500
- system administrator
  - SQL Servercompatibility 540
- system calls
  - from stored procedures 1326
  - xp\_cmdshell system procedure 1326
- system catalog 1356
  - about 1329
  - diagram 1330
  - list 1331
  - SQL Server compatibility 540
  - Transact-SQL 598
  - using 155
- system failure
  - recovery from 414
- system fuctions
  - xp\_msver 1327
- system functions 336, 573
  - tsequal 552

- system objects
  - and the dbo user 858, 876
- system procedures
  - about 1319
  - sa\_conn\_info 1321
  - sa\_conn\_properties 1321
  - sa\_db\_info 1321
  - sa\_db\_properties 1321
  - sa\_eng\_properties 1321
  - sp\_addgroup 601
  - sp\_addlogin 601
  - sp\_addmessage 601
  - sp\_addtype 601
  - sp\_adduser 601
  - sp\_changegroup 601
  - sp\_dboption 601
  - sp\_dropgroup 601
  - sp\_droplogin 601
  - sp\_dropmessage 601
  - sp\_droptype 601
  - sp\_dropuser 601
  - sp\_getmessage 601
  - sp\_helptext 601
  - sp\_password 601
  - Transact-SQL 601
  - xp\_cmdshell 1326
  - xp\_scanf 1328
  - xp\_sendmail 1324
  - xp\_sprintf 1327
  - xp\_startmail 1323
  - xp\_stopmail 1325
- SYSTEM statement
  - syntax 1179
- system tables
  - about 216, 1329
  - and character sets 358
  - and indexes 224
  - and integrity 244
  - and national languages 358
  - and views 222
  - diagram 1330
  - list 1331
  - permissions 397
  - SQL Server compatibility 540
  - SYSARTICLE 1331
  - SYSARTICLECOL 1332
  - SYSCATALOG 155
  - SYSCOLLATE 1332
  - SYSCOLLATION 358
  - SYSCOLPERM 1333
  - SYSCOLUMN 1334
  - SYSCOLUMNS 157
  - SYSDOMAIN 1335
  - SYSFILE 1336
  - SYSFKCOL 244, 1336
  - SYSFOREIGNKEY 244, 1337
  - SYSFOREIGNKEYS 244
  - SYSGROUP 1338
  - SYSICOL 224
  - SYSINDEX 224, 1338
  - SYSINDEXES 224
  - SYSINFO 358, 1339
  - SYSIXCOL 1340
  - SYSOPTION 1341
  - SYSPROCEDURE 1341
  - SYSROPCPARAM 1342
  - SYSROPCPERM 1343
  - SYS PUBLICATION 1344
  - SYSREMOTEUSER 1344
  - SYS SUBSCRIPTION 1346
  - SYSTABLE 158, 222, 244, 1346
  - SYSTABLEPERM 1077, 1348
  - system views 1355
  - SYSTRIGGER 244, 1349
  - SYSUSERMESSAGES 1351
  - SYSUSERPERM 1077, 1351
  - SYSUSERTYPE 1353
  - Transact-SQL 545, 598
  - Transact-SQL name conflicts 545
  - users and groups 397
  - using 155
- system variables 903
- system views
  - about 1355
  - and indexes 224
  - and integrity 244
  - and views 222
  - SYSCATALOG 1356
  - SYSCOLAUTH 1356
  - SYSCOLUMNS 1357
  - SYSFOREIGNKEYS 1357
  - SYSGROUPS 1358
  - SYSINDEXES 1358
  - SYSOPTIONS 1358
  - SYSROPCPARAMS 1359
  - SYSREMOTEUSERS 1359
  - SYSTABAUTH 1360
  - SYSTRIGGERS 1360

SYSUSERAUTH 1361  
SYSUSERLIST 1361  
SYSUSEROPTIONS 1361  
SYSUSERPERMS 1362  
SYSVIEWS 222, 1362

## T

table constraints 1022  
table editor 52  
table list 78, 95  
    FROM clause 1074  
table naming 215  
table number 1347  
tables 378  
    about 182, 211, 976, 1020  
    adding 52  
    adding columns 52  
    adding keys to 214  
    alias 115  
    altering 212, 231, 235, 976  
    altering definition 978  
    and foreign keys 117, 183  
    and view permissions 221  
    base tables 1021  
    characteristics 182  
    CHECK conditions 228, 235, 237  
    checks 237  
    columns 199  
    constraints 200, 214, 227, 228, 235  
    correlation name 115  
    creating 52, 211, 229, 235, 587, 1020  
    creating a primary key with Sybase Central 53  
    creating, in Sybase Central 211  
    deleting 56  
    deleting all rows from 1180  
    designing 185, 199  
    dropping 56, 214, 1053  
    editing existing 53  
    editor 52  
    GLOBAL temporary 1021  
    group owners 388  
    in publications 470  
    list of 78, 95  
    loading 370, 1105  
    loading using ISQL 371  
    looking up 388  
    name prefixes 391  
    naming 898  
    naming, in SQL 215  
    owner 377  
    owners 391  
    permissions 377, 378  
    primary keys 117  
    qualified names 388, 391  
    renaming 213, 980  
    special 155  
    structure 212  
    table lists 898  
    temporary 587, 1021, 1043  
    Transact-SQL 587  
    truncating 1180  
    unloading 366, 1182  
    using qualified names 309  
    validating 1187  
    viewing 50  
Taiwanese character set 352  
TAN function 940  
TaskSwitch property 955  
TaskSwitchCheck property 955  
temporary file  
    location 812  
temporary tables  
    and importing data 372  
    and query processing 332  
    and result sets 624  
    created 372  
    creating 1020  
    declared 372, 587  
    declaring 284, 1043  
    GLOBAL 1020  
    in procedures 624  
    LOCAL 1043  
    Transact-SQL 587  
text and image functions 572, 573  
TEXT data type  
    Transact-SQL 550  
TEXT file format 1167  
textptr function 572, 573  
THEN  
    IF expression 906  
thread count  
    database option 1155  
THREAD\_COUNT option 1150  
threads  
    and Embedded SQL 699  
three valued logic

- NULL value 1110
- three-valued logic 914
- time 922
- TIME data type 922
  - Transact-SQL 552
- time format
  - database option 1156
- TIME\_FORMAT option 1150
- times
  - comparing 925
- timestamp
  - adding a timestamp column to a table 553
  - and OmniCONNECT 650
  - timestamp column 552
    - data type 553
  - timestamp columns 1159
- TIMESTAMP data type 668, 922
  - Transact-SQL 552
- timestamp format
  - database option 1156
- TINYINT data type 921
  - Transact-SQL 549
- TMP environment variable 812
- TODAY function 949, 1331
- TRACEBACK function 302, 967
- trademark information
  - retrieving 1327
- trailing blanks in strings 544
- transaction log
  - about 42, 403
  - allocating space for 209, 972
  - and database performance 320
  - and portable computers 262, 263
  - and primary keys 404
  - and replication 427
  - and SQL Remote 427, 863
  - and uncommitted changes 418
  - DBLOG 871
  - erasing 835
  - Message Agent 863
  - mirror 406, 408
  - mirroring 403
  - monitoring performance 319
  - options 413
  - preallocating space for 972
  - size 404
  - Transact-SQL DUMP DATABASE 588
  - Transact-SQL LOAD DATABASE 588
  - translation utilities 850
  - TRUNCATE TABLE statement 1180
  - updating databases from 262
  - utilities 871
- transaction log mirror
  - about 406
  - and replication 503
  - and SQL Remote 498
  - purpose 406
  - requirements 406
- transaction management 582, 584, 592
  - in Transact-SQL 582, 584, 592
- transaction modes
  - chained 582, 1160
  - unchained 582, 1160
- transaction processing 246, 249
  - and performance 249
- transactions
  - and concurrency 247
  - and data recovery 247
  - and deadlock 254
  - and lost updates 256
  - and procedures 308
  - and triggers 308
  - blocking 254
  - closing cursors 1160
  - committing 993
  - coordinating with multiple servers 260
  - overview 246
  - ROLLBACK statement 1137
  - ROLLBACK TO SAVEPOINT statement 1138
  - SAVEPOINT statement 1140
  - serializable 257
- Transact-SQL
  - about 536
  - aggregate functions 565
  - ALL conditions 578
  - ALLOW\_NULLS\_BY\_DEFAULT option 546, 1158
  - and logical operators 579
  - and procedures 626
  - and Watcom-SQL 606
  - ANY conditions 578
  - arithmetic operators 563
  - assigning values to variables 626
  - AUTOMATIC\_TIMESTAMP option 1159
  - batches 613
  - BETWEEN conditions 577

- binary data type compatibility 551
- bit data type compatibility 551
- bitwise operators 564
- case sensitivity
  - user-defined data types 547
- catalog procedures 601, 602
- character data types compatibility 550
- column NULLs compatibility 1158, 1161
- COMMIT 584
- comparison conditions 576
- comparison operators 577
- configuring databases for 544
- constants 561
- CREATE INDEX statement 585
- CREATE MESSAGE 586
- CREATE PROCEDURE statement 615
- CREATE SCHEMA statement 587, 1017
- CREATE TABLE statement 587
- data type compatibility 549
- data type conversion functions 570
- database options 594
- date and time data type compatibility 552
- date and time functions 568
- dbo user 545
- decimal data type compatibility 550
- DECLARE section 614
- delete permissions 1159
- DELETE statement 589
- error handling in 620
- executing stored procedures 617
- EXISTS conditions 578
- expressions 561
- functions 565
- global variables 557
- GRANT statement 589
- hexadecimal constants 1162
- identity column 554
- IN conditions 578
- INSERT statement 590
- integer data type compatibility 549
- IS NULL conditions 578
- joins 591
- LIKE conditions 577
- local variables 556
- money data type compatibility 551
- NULL behavior 1159
- numeric functions 565
- operators 563
  - outer join operators 564
- procedures 610, 615
- QUOTED\_IDENTIFIER option 546, 1161
- READTEXT statement 591
- REVOKE statement 589
- ROLLBACK statement 592
- search conditions 576, 577, 578, 579
- SELECT statement 592
- SET statement 594
- string concatenation operator 563
- string functions 566
- strings 562
- system catalog 598
- system functions 573
- system procedures 601
- system tables 545
- text and image functions 572, 573
- timestamp column 552
- triggers 611
  - update permissions 1159
- UPDATE statement 596
- user-defined data types 555
- viewing procedures in 58
- WRITETEXT statement 597
- writing portable SQL 543
- translating
  - procedures 608
- tree
  - and indexes 327
- TriggerPages property 959
- triggers
  - about 266
  - allowed statements in 286
  - altering 982
  - and control statements 283
  - and permissions 385
  - and referential integrity actions 1162
  - and replication 479, 1160
  - and SQL Remote 479, 490, 491
  - and Transact-SQL compatibility 611
  - benefits of 267
  - command delimiter and 309
  - creating 277
  - cursors in 295
  - dropping 279, 1053
  - error handling 300, 304
  - errors in 300
  - executing 279
  - execution permissions 279, 385

- permissions for creating 377
- replicating 490, 491
- RESOLVE UPDATE 507, 509, 510
- ROLLBACK statement 1139
- statement-level 611
- Transact-SQL 616
- TRUNCATE TABLE statement 1180
- uniqueness of names 547
- using 265
- using ON EXCEPTION RESUME 300
- viewing 57
- warnings in 300
- Watcom-SQL 982, 1028
- TRIM function 944
- TRUE condition 913
- TRUNCATE function 940
- TRUNCATE TABLE statement
  - syntax 1180
- truncating a table 1180
- truncation
  - of character strings 1162
  - on FETCH 672
- truncation length
  - ISQL option 1169
- tsequal function 552
- tsequal SQL Server function 573
- TSQL\_HEX\_CONSTANT option 1162
- TSQL\_VARIABLES option 1163
- tuples 182
- Tutorial
  - using Sybase Central 45
- Tutorials
  - Sybase Central 45
- two-phase commit 260
  - and Open Server Gateway 648
  - PREPARE TO COMMIT statement 1123
- type conversions 929
- types
  - about data types 917

## U

- UCASE function 944
- unchained transaction mode 582, 1160
- UncommitOp property 955
- unicode collation 352
- UNION operation 1181
- unique

- constraint 1022
- unique indexes 1007
- uniqueness of object names 547
- Unix 809
- Unix commands 809
- UNKNOWN condition 906, 913
- UNLOAD TABLE statement
  - about 366
  - syntax 1182
- unloading databases
  - and replication 505
- unloading tables 366, 1182
- UnschReq property 959
- UP ISQL command 848
- UPDATE
  - IF UPDATE clause 616, 982, 1028
- UPDATE (positioned) statement
  - syntax 1186
- update column permission 1333
- UPDATE permissions 382
- UPDATE statement
  - and integrity 227, 243
  - and transaction logs 263
  - examples 129
  - syntax 1183
  - Transact-SQL 596
  - truncation of strings 1162
- updates
  - and joins 1183, 1184
  - and views 1186
  - ANSI behavior 1159
  - applying 262
  - Transact-SQL permissions 1159
- upgrade utility 7
- upgrading
  - and replication 505
  - and SQL Remote 505
  - databases 882, 883
  - SQL Remote installations 494
- upgrading a database 7
- used\_pgs SQL Server function 573
- USER
  - default 1021
  - number 1351
  - special constant 900, 1331
- user IDs
  - and connecting 162
  - and permissions 140
  - and system tables 156



- benefits of different user IDs 376
- changing passwords 1086
- creating 380
- creation 1086
- default 232
- granting permissions to 394
- in the system tables 1347, 1351
- system views 1361
- user\_id SQL Server function 573
- user\_name SQL Server function 573
- user-defined data types
  - about 927
  - case-sensitivity 547
  - CHECK conditions 236, 238
  - CREATE DATATYPE statement 1002
  - dropping 1053
  - Transact-SQL 555
- user-defined functions
  - calling 274
  - CREATE FUNCTION statement 1005
  - creating 273
  - dropping 274
  - RETURN statement 1130
  - using 273
- userid 84
  - about 898
  - defined 163
  - using with ISQL 70
- Userid property 955
- users
  - adding 61
  - adding to groups, using Sybase Central 61, 387
  - creating 61, 380
  - creating individual 380
  - creating, in Sybase Central 380
  - deleting 385
  - dropping 1133
  - in SQL Server and SQL Anywhere 541, 589
  - inspecting 397
  - managing 60, 375
  - occasionally connected 262
  - remote 426
- UTF8 collation 352
- utilities *See* database utilities
  - backup 824
  - collation 829
  - compression 832
  - erase 834

- information 837
- initialization 839
- ISQL 846
- log translation 849
- Open Server Gateway 853
- Open Server information 855
- Open Server stop 856
- remote database extraction 858
- remote message ageng 863
- SQL preprocessor 892
- stop 857, 867
- transaction log 869
- uncompression 873
- unload 876
- upgrade 882
- validation 885
- write file 888

## V

- valid\_name SQL Server function 573
- valid\_user SQL Server function 573
- VALIDATE TABLE statement
  - syntax 1187
- validating databases 886
- validating tables 1187
- validity checking 133, 214
- VARBINARY data type
  - Transact-SQL 551
- VARCHAR data type 668, 918
  - Transact-SQL 550
- variable result sets
  - from procedures 294, 1010, 1051, 1121
- variable result sets from procedures 294
- variables 901
  - about 898, 1031
  - assigning values to 594, 626
  - declaring 284
  - dropping 1060
  - global 556, 557, 903
  - in procedures 626
  - local 556, 594
  - obtaining values using SELECT 560
  - SELECT statement and 594, 626
  - SET statement 594, 626
  - SET VARIABLE statement 1145
  - setting values of 290
- VERIFY\_ALL\_COLUMNS

- replication option 1164
- VERIFY\_ALL\_COLUMNS option 510
- VERIFY\_THRESHOLD
  - replication option 1164
- version number
  - retrieving 1327
- ViewPages property 959
- views
  - about 137, 983, 1033
  - altering 983
  - and indexes 1007
  - and security 393
  - and table permissions 221
  - and tables 217, 219
  - and updates 1186
  - check option 219
  - creating 138, 217, 1033
  - creating, in Sybase Central 218
  - deleting 139, 221
  - dropping 1053
  - dropping, in Sybase Central 222
  - examples 1355
  - modifying 221
  - owner 377
  - permissions 140, 221, 377, 378
  - system views 1362
  - updatable 219
  - update permissions 394
  - updating 219
  - using 218
  - using indexes with 223
  - viewing 57
  - working with 217
- VIM 458
- VIM link 429
- VIM message type
  - about 976
- VIM messages
  - about 1015, 1058
- Visual C++
  - support for 656
- VoluntaryBlock property 955, 959
- VX-Rexx 802

## W

- WAIT\_FOR\_COMMIT option 1150
- WaitReadCmp property 955, 959

- WaitReadOpt property 955, 959
- WaitReadSys property 955, 959
- WaitReadTemp property 955, 959
- WaitReadUnkn property 955, 959
- warm links 767
- warnings
  - error messages 1210
  - in procedures 300
  - in triggers 300
  - SQLSTATE values 1201
- WATCOM C/C++
  - support for 656
- Watcom-SQL
  - and Transact-SQL 606
  - new features in 18
- WATFILE file format 1165, 1167
- WEEKS function 949
- WHENEVER statement
  - syntax 1188
- WHERE clause 132
  - and BETWEEN conditions 110
  - and date comparisons 106
  - and GROUP BY clause 125
  - and ORDER BY clause 105
  - and pattern matching 108
  - examples 105
  - in publications 472
  - SELECT statement 1143
  - UPDATE statement 129
- WHILE statement
  - control statements 283
  - syntax 1108
  - Transact-SQL 622
- wide fetches 686
- wide inserts 686, 1062
- wide puts 686, 1125
- window
  - moving left or right 75
- Windows 3.x
  - and SQL Remote 457
  - background processing 716
- Windows 95
  - and ISQL 67
  - and SQL Remote 457
  - DOS client applications 32, 33
  - DOS client applications on 32
  - Windows 3.x client applications 32, 33
  - Windows 3.x client applications on 32
- Windows DLL

- for Embedded SQL 701
- Windows NT
  - and ISQL 67
  - and SQL Remote 457
  - DOS client applications 32, 33
  - services 518
  - Windows 3.x client applications 32, 33
- Windows NT DLL
  - for Embedded SQL 701
- WITH GRANT OPTION clause 383
- WITH HOLD clause
  - OPEN statement 1112
- Wizards
  - in Sybase Central 63
- WOD50T.DLL 170
- WOD50W.DLL 170
- write files 42, 890
  - and deployment 263
  - erasing 835
- write locks 250
- WRITETEXT statement
  - Transact-SQL 597
- WSQL HLI
  - about 654

## X

- xp\_cmdshell system procedure 1326
- xp\_msver system function 1327
- xp\_scanf system procedure 1328
- xp\_sendmail system procedure 1324
- xp\_sprintf system procedure 1327
- xp\_startmail system procedure 1323
- xp\_stopmail system procedure 1325

## Y

- Y2K 930
- year 2000 930
  - NEAREST\_CENTURY option 1160
- YEAR function 950
- YEARS function 950
- YMD function 950