

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

Alphabetical list of SQL statements

There are many SQL statements supported by the software.

In this section:

- **ALLOCATE DESCRIPTOR statement [ESQL]**
Allocates space for a SQL descriptor area (SQLDA).
- **ALTER DATABASE statement**
Upgrades the database, turns jConnect support for a database on or off, calibrates the database, changes the transaction log and transaction log mirror file names, or forces a mirror server to take ownership of a database.
- **ALTER DBSPACE statement**
Preallocates space for a dbspace or for the transaction log, or updates the catalog when a dbspace file is renamed or moved.
- **ALTER DOMAIN statement**
Renames a user-defined domain or data type.
- **ALTER EVENT statement**
Changes the definition of an event or its associated handler for automating predefined actions, or alters the definition of scheduled actions. You can also use this statement to hide the definition of an event handler.
- **ALTER EXTERNAL ENVIRONMENT statement**
Specifies the location of an external environment such as Java, PHP, or Perl.
- **ALTER FUNCTION statement**
Modifies a function.
- **ALTER INDEX statement**
Renames an index, primary key, or foreign key, or changes the clustered nature of an index.
- **ALTER LDAP SERVER statement**
Alters an LDAP server configuration object.
- **ALTER LOGIN POLICY statement**
Alters an existing login policy.
- **ALTER MATERIALIZED VIEW statement**
Alters a materialized view.
- **ALTER MIRROR SERVER statement**
Modifies the attributes of a mirror server.
- **ALTER ODATA PRODUCER statement**
Alters an OData Producer.
- **ALTER PROCEDURE statement**
Modifies a procedure.
- **ALTER PUBLICATION statement [MobiLink] [SQL Remote]**
Alters a publication. In MobiLink, a publication identifies synchronized data in a SQL Anywhere remote database. In SQL Remote, a publication identifies replicated data in both consolidated and remote databases.
- **ALTER REMOTE MESSAGE TYPE statement [SQL Remote]**
Changes the publisher's message system, or the publisher's address for a given message

system, for a message type that has been created.

- **ALTER ROLE statement**
Migrates a compatibility role to a user-defined role, and then drops the compatibility role.
- **ALTER SEQUENCE statement**
Alters a sequence.
- **ALTER SERVER statement**
Modifies the attributes of a remote server.
- **ALTER SERVICE statement [HTTP web service]**
Alters an existing HTTP web service.
- **ALTER SERVICE statement [SOAP web service]**
Alters an existing SOAP or DISH service over HTTP.
- **ALTER SPATIAL REFERENCE SYSTEM statement**
Changes the settings of an existing spatial reference system. See the Remarks section for considerations before altering a spatial reference system.
- **ALTER STATISTICS statement**
Controls whether statistics are automatically updated on a column, or columns, in a table.
- **ALTER SYNCHRONIZATION PROFILE statement [MobiLink]**
Changes a SQL Anywhere synchronization profile. Synchronization profiles are named collections of synchronization options that can be used to control synchronization.
- **ALTER SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]**
Alters the properties of a synchronization subscription in a SQL Anywhere remote database.
- **ALTER SYNCHRONIZATION USER statement [MobiLink]**
Alters the properties of a MobiLink user in a SQL Anywhere remote database.
- **ALTER TABLE statement**
Modifies a table definition or disables dependent views.
- **ALTER TEXT CONFIGURATION statement**
Alters a text configuration object.
- **ALTER TEXT INDEX statement**
Alters the definition of a text index.
- **ALTER TIME ZONE statement**
Modifies a time zone object.
- **ALTER TRACE EVENT SESSION statement**
Adds or removes trace events from a session, adds or removes targets from a session, or starts or stops a trace session.
- **ALTER TRIGGER statement**
Replaces a trigger definition with a modified version. You must include the entire new trigger definition in the ALTER TRIGGER statement.
- **ALTER USER statement**
Alters user settings.
- **ALTER VIEW statement**
Replaces a view definition with a modified version.
- **ATTACH TRACING statement (deprecated)**
The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database. The ATTACH TRACING statement starts a diagnostic tracing session (starts sending diagnostic information to the diagnostic tables).

- **BACKUP DATABASE statement**
Backs up a database and transaction log.
- **BEGIN SNAPSHOT statement**
Starts a snapshot at a specified period in time for use with snapshot isolation transactions.
- **BEGIN statement**
Specifies a compound statement.
- **BEGIN statement [TSQL]**
Specifies a compound statement.
- **BEGIN TRANSACTION statement [T-SQL]**
Begins a user-defined transaction.
- **BREAK statement [T-SQL]**
Exits a compound statement or loop.
- **CALL statement**
Invokes a procedure.
- **CASE statement**
Selects an execution path based on multiple cases.
- **CASE statement [T-SQL]**
Selects an execution path based on multiple cases.
- **CHECKPOINT statement**
Checkpoints the database.
- **CLEAR statement [Interactive SQL]**
Closes any open result sets in Interactive SQL.
- **CLOSE statement [ESQL] [SP]**
Closes a cursor.
- **COMMENT statement**
Stores a comment for a database object in the system tables.
- **COMMIT statement**
Makes changes to the database permanent, or terminates a user-defined transaction.
- **CONFIGURE statement [Interactive SQL]**
Opens the Interactive SQL Options window.
- **CONNECT statement [ESQL] [Interactive SQL]**
Establishes a connection to a database.
- **CONTINUE statement**
Restarts a loop.
- **CREATE CERTIFICATE statement**
Adds or replaces a certificate in the database from the given file or string. To create a certificate, use the Certificate Creation utility (createcert).
- **CREATE DATABASE statement**
Creates a database.
- **CREATE DBSPACE statement**
Defines a new database space and creates the associated database file.
- **CREATE DECRYPTED DATABASE statement**
Creates a decrypted copy of an existing database, including all transaction logs and dbspaces.
- **CREATE DECRYPTED FILE statement**

Creates a decrypted copy of a strongly encrypted database, and can be used to create decrypted copies of transaction logs, transaction log mirrors, and dbspaces.

- **CREATE DOMAIN statement**
Creates a domain in a database.
- **CREATE ENCRYPTED DATABASE statement**
Creates an encrypted copy of an existing database, including all transaction logs and dbspaces; or creates a copy of an existing database with table encryption enabled.
- **CREATE ENCRYPTED FILE statement**
Creates a strongly encrypted copy of a database file, transaction log, transaction log mirror, or dbspace.
- **CREATE EVENT statement**
Defines an event and its associated handler for automating predefined actions, and to define scheduled actions.
- **CREATE EXISTING TABLE statement**
Creates a new proxy table, which represents an existing object on a remote server.
- **CREATE EXTERNLOGIN statement**
Assigns an alternate login name and password to be used when communicating with a remote server.
- **CREATE FUNCTION statement [External call]**
Creates an interface to a native or external function.
- **CREATE FUNCTION statement [Web service]**
Creates a web client function that makes an HTTP or SOAP over HTTP request.
- **CREATE FUNCTION statement**
Creates a user-defined SQL function in the database.
- **CREATE INDEX statement**
Creates an index on a specified table or materialized view.
- **CREATE LDAP SERVER statement**
Creates an LDAP server configuration object.
- **CREATE LOCAL TEMPORARY TABLE statement**
Creates a local temporary table within a procedure that persists after the procedure completes and until it is either explicitly dropped, or until the connection terminates.
- **CREATE LOGIN POLICY statement**
Creates a login policy.
- **CREATE MATERIALIZED VIEW statement**
Creates a materialized view.
- **CREATE MESSAGE statement [T-SQL]**
Creates a message number/message string pair. The message number can be used in PRINT and RAISERROR statements.
- **CREATE MIRROR SERVER statement**
Creates or replaces a mirror server that is being used for database mirroring or read-only scale-out.
- **CREATE MUTEX statement**
Creates or replaces a mutex (lock) that can be used to lock a resource such as a file or a procedure.
- **CREATE ODATA PRODUCER statement**

Creates or replaces an OData Producer.

- **CREATE PROCEDURE statement [External call]**
Creates an interface to a native or external procedure.
- **CREATE PROCEDURE statement [Web service]**
Creates a user-defined web client procedure that makes HTTP or SOAP requests to an HTTP server.
- **CREATE PROCEDURE statement [T-SQL]**
Creates a new procedure in the database in a manner compatible with Adaptive Server Enterprise.
- **CREATE PROCEDURE statement**
Creates a user-defined SQL procedure in the database.
- **CREATE PUBLICATION statement [MobiLink] [SQL Remote]**
Creates a publication. In MobiLink, a publication identifies synchronized data in a SQL Anywhere remote database. In SQL Remote, publications identify replicated data in both consolidated and remote databases.
- **CREATE REMOTE [MESSAGE] TYPE statement [SQL Remote]**
Identifies a message link and return address for outgoing messages from a database.
- **CREATE ROLE statement**
Creates or replaces a role, creates a user-extended role, or modifies administrators for a role.
- **CREATE SCHEMA statement**
Creates a collection of tables and views for a database user.
- **CREATE SEMAPHORE statement**
Creates or replaces a semaphore and establishes the initial value for its counter. A semaphore is a locking mechanism that uses a counter to communicate and control the availability of a resource such as an external library or procedure.
- **CREATE SEQUENCE statement**
Creates a sequence that can be used to generate primary key values that are unique across multiple tables, and for generating default values for a table.
- **CREATE SERVER statement**
Creates a remote server or a directory access server.
- **CREATE SERVICE statement [HTTP web service]**
Creates a new HTTP web service.
- **CREATE SERVICE statement [SOAP web service]**
Creates a new SOAP over HTTP or DISH service.
- **CREATE SPATIAL REFERENCE SYSTEM statement**
Creates or replaces a spatial reference system.
- **CREATE SPATIAL UNIT OF MEASURE statement**
Creates or replaces a spatial unit of measurement.
- **CREATE STATISTICS statement**
Recreates the column statistics used by the optimizer, and stores them in the ISYSCOLSTAT system table.
- **CREATE SUBSCRIPTION statement [SQL Remote]**
Creates a subscription for a user to a publication.
- **CREATE SYNCHRONIZATION PROFILE statement [MobiLink]**
Creates a SQL Anywhere synchronization profile.

- **CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]**
Creates a subscription in a SQL Anywhere remote database between a MobiLink user and a publication.
- **CREATE SYNCHRONIZATION USER statement [MobiLink]**
Creates a MobiLink user in a SQL Anywhere remote database.
- **CREATE TABLE statement**
Creates a new table in the database and, optionally, creates a table on a remote server.
- **CREATE TEMPORARY TRACE EVENT statement**
Creates a user trace event that persists until the database is stopped.
- **CREATE TEMPORARY TRACE EVENT SESSION statement**
Creates a user trace event session.
- **CREATE TEXT CONFIGURATION statement**
Creates a text configuration object for use with building and updating text indexes.
- **CREATE TEXT INDEX statement**
Creates a text index.
- **CREATE TIME ZONE statement**
Creates a simulated time zone that can be used by any database.
- **CREATE TRIGGER statement**
Creates a trigger on a table.
- **CREATE TRIGGER statement [T-SQL]**
Creates a new trigger in the database in a manner compatible with Adaptive Server Enterprise.
- **CREATE USER statement**
Creates a database user or group.
- **CREATE VARIABLE statement**
Creates a connection- or database-scope variable.
- **CREATE VIEW statement**
Creates a view on the database.
- **DEALLOCATE DESCRIPTOR statement [ESQL]**
Frees memory associated with a SQL descriptor area.
- **DEALLOCATE statement**
This statement has no effect in SQL Anywhere, and is ignored. It is provided for compatibility with Adaptive Server Enterprise and Microsoft SQL Server. Refer to your Adaptive Server Enterprise or Microsoft SQL Server documentation for more information about this statement.
- **Declaration section [ESQL]**
Declares host variables in an Embedded SQL program. Host variables are used to exchange data with the database.
- **DECLARE CURSOR statement [ESQL] [SP]**
Declares a cursor.
- **DECLARE LOCAL TEMPORARY TABLE statement**
Declares a local temporary table.
- **DECLARE statement**
Declares a SQL variable (connection-scope) or an exception within a compound statement (BEGIN...END).
- **DELETE statement (positioned) [ESQL] [SP]**
Deletes the data at the current location of a cursor.

- **DELETE statement**
Deletes rows from the database.
- **DESCRIBE statement [ESQL]**
Gets information about the host variables required to store data retrieved from the database, or host variables required to pass data to the database.
- **DESCRIBE statement [Interactive SQL]**
Returns information about a given database object.
- **DETACH TRACING statement (deprecated)**
The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database. The DETACH TRACING session ends a diagnostic tracing session.
- **DISCONNECT statement [ESQL] [Interactive SQL]**
Drops a connection to a database.
- **DROP CERTIFICATE statement**
Drops a certificate from the database.
- **DROP CONNECTION statement**
Drops a user's connection to the database.
- **DROP DATABASE statement**
Deletes all database files associated with a database.
- **DROP DATATYPE statement**
Removes a data type from the database.
- **DROP DBSPACE statement**
Removes a dbspace from the database.
- **DROP DOMAIN statement**
Removes a domain (data type) from the database.
- **DROP EVENT statement**
Drops an event from the database.
- **DROP EXTERNLOGIN statement**
Drops an external login from the database.
- **DROP FUNCTION statement**
Removes a function from the database.
- **DROP INDEX statement**
Removes an index from the database.
- **DROP LDAP SERVER statement**
Drops an LDAP server configuration object.
- **DROP LOGIN POLICY statement**
Drops a login policy.
- **DROP MATERIALIZED VIEW statement**
Removes a materialized view from the database.
- **DROP MESSAGE statement**
Removes a message from the database.
- **DROP MIRROR SERVER statement**
Drops a mirror server.
- **DROP MUTEX statement**
Drops the specified mutex.

- **DROP ODATA PRODUCER statement**
Drops an OData Producer.
- **DROP PROCEDURE statement**
Removes a procedure from the database.
- **DROP PUBLICATION statement [MobiLink] [SQL Remote]**
Drops a publication.
- **DROP REMOTE MESSAGE TYPE statement [SQL Remote]**
Deletes a message type definition from a database.
- **DROP REMOTE CONNECTION statement**
Drops remote data access connections to a remote server.
- **DROP ROLE statement**
Removes a role from the database, or converts a user-extended role back to a regular user.
- **DROP SEMAPHORE statement**
Drops a semaphore.
- **DROP SEQUENCE statement**
Drops a sequence.
- **DROP SERVER statement**
Drops a remote server from the catalog.
- **DROP SERVICE statement**
Drops a web service.
- **DROP SPATIAL REFERENCE SYSTEM statement**
Drops a spatial reference system.
- **DROP SPATIAL UNIT OF MEASURE statement**
Drops a spatial unit of measurement.
- **DROP STATEMENT statement [ESQL]**
Frees statement resources.
- **DROP STATISTICS statement**
Erases all column statistics on the specified columns.
- **DROP SUBSCRIPTION statement [SQL Remote]**
Drops a subscription for a user from a publication.
- **DROP SYNCHRONIZATION PROFILE statement [MobiLink]**
Deletes a SQL Anywhere synchronization profile.
- **DROP SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]**
Drops a synchronization subscription in a remote database.
- **DROP SYNCHRONIZATION USER statement [MobiLink]**
Drops one or more synchronization users from a SQL Anywhere remote database.
- **DROP TABLE statement**
Removes a table from the database.
- **DROP TEXT CONFIGURATION statement**
Drops a text configuration object.
- **DROP TEXT INDEX statement**
Removes a text index from the database.
- **DROP TIME ZONE statement**
Drops a time zone from the database.

- **DROP TRACE EVENT statement**
Drops a user-defined trace event.
- **DROP TRACE EVENT SESSION statement**
Drops a trace event session.
- **DROP TRIGGER statement**
Removes a trigger from the database.
- **DROP USER statement**
Drops a user.
- **DROP VARIABLE statement**
Drops a SQL variable.
- **DROP VIEW statement**
Removes a view from the database.
- **EXCEPT statement**
Returns the set difference of two query blocks.
- **EXECUTE IMMEDIATE statement [SP]**
Enables dynamically constructed statements to be executed from within a procedure.
- **EXECUTE statement [ESQL]**
Executes a prepared SQL statement.
- **EXECUTE statement [T-SQL]**
Invokes a procedure (an Adaptive Server Enterprise-compatible alternative to the CALL statement), executes a prepared SQL statement in Transact-SQL.
- **EXIT statement [Interactive SQL]**
Leaves Interactive SQL.
- **EXPLAIN statement [ESQL]**
Retrieves a text specification of the optimization strategy used for a particular cursor.
- **FETCH statement [ESQL] [SP]**
Positions, or re-positions, a cursor to a specific row, and then copies expression values from that row into variables accessible from within the stored procedure or application.
- **FOR statement**
Repeats the execution of a statement list once for each row in a cursor.
- **FORWARD TO statement**
Sends native syntax SQL statements to a remote server.
- **FROM clause**
Specifies the database tables or views involved in a DELETE, SELECT, or UPDATE statement. When used within a SELECT statement, the FROM clause can also be used in a MERGE or INSERT statement.
- **GET DATA statement [ESQL]**
Gets string or binary data for one column of the current row of a cursor.
- **GET DESCRIPTOR statement [ESQL]**
Retrieves information about a variable within a descriptor area, or retrieves its value.
- **GET OPTION statement [ESQL]**
Gets the current setting of an option. It is recommended that you use the CONNECTION_PROPERTY function instead.
- **GOTO statement**
Branches to a labeled statement.

- **GRANT statement**
Grant system and object-level privileges to users and roles.
- **GRANT ROLE statement**
Grant roles to users and roles.
- **GRANT CONNECT statement**
Creates a new user, and can also be used by a user to change their own password. However, it is recommended that you use the CREATE USER statement to create users instead of the GRANT CONNECT statement.
- **GRANT CONSOLIDATE statement [SQL Remote]**
Identifies the database immediately above the current database in a SQL Remote hierarchy, that will receive messages from the current database.
- **GRANT CREATE statement**
Allows a user to create database objects in the specified dbspace.
- **GRANT EXECUTE statement**
Grants a user privilege to execute a procedure or user-defined function.
- **GRANT INTEGRATED LOGIN statement**
Grants a user privilege to execute a procedure or user-defined function.
- **GRANT KERBEROS LOGIN statement**
Creates a Kerberos authenticated login mapping from one or more Kerberos principals to an existing database user ID.
- **GRANT PUBLISH statement [SQL Remote]**
Grants publisher rights to a user ID. You must have the SYS_REPLICATION_ADMIN_ROLE system role to grant publisher rights.
- **GRANT REMOTE statement [SQL Remote]**
Identifies a database immediately below the current database in a SQL Remote hierarchy, that will receive messages from the current database. These are called remote users.
- **GRANT USAGE ON SEQUENCE statement**
Grants privilege to use a specified sequence.
- **GROUP BY clause**
Groups columns, alias names, and functions as part of the SELECT statement.
- **HELP statement [Interactive SQL]**
Provides help in the Interactive SQL environment.
- **IF statement**
Controls conditional execution of SQL statements.
- **IF statement [T-SQL]**
Controls conditional execution of a SQL statement, as an alternative to the Watcom SQL IF statement.
- **INCLUDE statement [ESQL]**
Includes a file into a source program to be scanned by the SQL preprocessor.
- **INPUT statement [Interactive SQL]**
Imports data into a database table from an external file, from the keyboard, from an ODBC data source, or from a shapefile.
- **INSERT statement**
Inserts a single row or a selection of rows from elsewhere in the database into a table.
- **INSTALL EXTERNAL OBJECT statement**

Installs an object that can be run in an external environment.

- **INSTALL JAVA statement**
Makes Java classes available for use within a database.
- **INTERSECT statement**
Computes the intersection between the result sets of two or more queries.
- **LEAVE statement**
Leaves a compound statement or loop.
- **LOAD STATISTICS statement**
Intended for internal use, this statement is used by the dbunload utility to unload column statistics from the old database into the ISYSCOLSTAT system table.
- **LOAD TABLE statement**
Imports bulk data into a database table from an external file.
- **LOCK FEATURE statement**
Prevents other concurrent connections from using a database server feature.
- **LOCK MUTEX statement**
Locks a resource such as a file or system procedure using a predefined mutex.
- **LOCK TABLE statement**
Prevents other concurrent transactions from accessing or modifying a table.
- **LOOP statement**
Repeats the execution of a statement list.
- **MERGE statement**
Merges tables, views, and procedure results into a table or view.
- **MESSAGE statement**
Displays a message.
- **NOTIFY SEMAPHORE statement**
Increments the counter associated with a semaphore.
- **NOTIFY TRACE EVENT statement**
Logs a user-defined trace event to a trace session.
- **OPEN statement [ESQL] [SP]**
Opens a previously declared cursor to access information from the database.
- **OUTPUT statement [Interactive SQL]**
Outputs the current query results to a file.
- **PARAMETERS statement [Interactive SQL]**
Specifies parameters to an Interactive SQL script file.
- **PASSTHROUGH statement [SQL Remote]**
Starts or stops passthrough mode for SQL Remote administration.
- **PIVOT clause**
Pivots a table expression in the FROM clause of a SELECT statement (FROM *pivoted-derived-table*) into a pivoted derived table. Pivoted derived tables offer an easy way to rotate row values from a column in a table expression into multiple columns and perform aggregation where needed on the columns included in the result set.
- **PREPARE statement [ESQL]**
Prepares a statement to be executed later, or defines a cursor.
- **PREPARE TO COMMIT statement**
Checks whether a COMMIT can be performed successfully.

- **PRINT statement [T-SQL]**
Returns a message to the client, or display a message in the database server messages window.
- **PUT statement [ESQL]**
Inserts a row into the specified cursor.
- **RAISERROR statement**
Signals an error and sends a message to the client.
- **READ statement [Interactive SQL]**
Reads Interactive SQL statements from a file.
- **READTEXT statement [T-SQL]**
Reads text and image values from the database, starting from a specified offset and reading a specified number of bytes. This feature is provided solely for compatibility with Transact-SQL and its use is not recommended.
- **REFRESH MATERIALIZED VIEW statement**
Initializes or refreshes the data in a materialized view by executing its query definition.
- **REFRESH TEXT INDEX statement**
Refreshes a text index.
- **REFRESH TRACING LEVEL statement (deprecated)**
The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database. The REFRESH TRACING LEVEL statement reloads the tracing levels from the sa_diagnostic_tracing_level table while a tracing session is in progress.
- **RELEASE MUTEX statement**
Releases the specified connection-scope mutex, if it is locked by the current connection.
- **RELEASE SAVEPOINT statement**
Releases a savepoint within the current transaction.
- **REMOTE RESET statement [SQL Remote]**
Starts all subscriptions for a remote user in a single transaction in custom database-extraction procedures.
- **REMOVE EXTERNAL OBJECT statement**
Removes an external object from the database.
- **REMOVE JAVA statement**
Removes a class or JAR file from a database.
- **REORGANIZE TABLE statement**
Defragments tables when a full rebuild of the database is not possible due to the requirements for continuous access to the database.
- **RESIGNAL statement [SP]**
Resignals an exception condition.
- **RESTORE DATABASE statement**
Restores a backed up database from an archive.
- **RESUME statement**
Resumes execution of a cursor that returns result sets.
- **RETURN statement**
Exits from a function, procedure, or batch unconditionally, optionally providing a return value.
- **REVOKE CONSOLIDATE statement [SQL Remote]**
Stops a consolidated database from receiving SQL Remote messages from this database.

- **REVOKE PUBLISH statement [SQL Remote]**
Terminates the identification of the named user ID as the current publisher. You must have the SYS_REPLICATION_ADMIN_ROLE system role to revoke publisher rights.
- **REVOKE REMOTE statement [SQL Remote]**
Stops a user from being able to receive SQL Remote messages from this database.
- **REVOKE statement**
Revokes roles and privileges from users and roles.
- **REVOKE ROLE statement**
Revokes roles and privileges from users and roles.
- **ROLLBACK statement**
Ends a transaction and undo any changes made since the last COMMIT or ROLLBACK.
- **ROLLBACK TO SAVEPOINT statement**
Cancels any changes made since a SAVEPOINT.
- **ROLLBACK TRANSACTION statement [T-SQL]**
Cancels any changes made since a SAVE TRANSACTION.
- **ROLLBACK TRIGGER statement**
Undoes any changes made in a trigger.
- **SAVE TRANSACTION statement [T-SQL]**
Establishes a savepoint within the current transaction.
- **SAVEPOINT statement**
Establishes a savepoint within the current transaction.
- **SELECT statement**
Retrieves information from the database.
- **SET CONNECTION statement [Interactive SQL] [ESQL]**
Changes the active database connection.
- **SET DESCRIPTOR statement [ESQL]**
Describes the variables in a SQL descriptor area and to place data into the descriptor area.
- **SET MIRROR OPTION statement**
Changes the values of options that control the settings for database mirroring and read-only scale-out.
- **SET OPTION statement**
Changes the values of database and connection options.
- **SET OPTION statement [Interactive SQL]**
Changes the values of Interactive SQL options.
- **SET REMOTE OPTION statement [SQL Remote]**
Sets a message control parameter for a SQL Remote message link.
- **SET SQLCA statement [ESQL]**
Instructs the SQL preprocessor to use a SQLCA other than the default, global *sqlca*.
- **SET statement**
Assigns a value to a SQL variable.
- **SET statement [T-SQL]**
Sets database options for the current connection in an Adaptive Server Enterprise-compatible manner.
- **SETUSER statement**
Allows a user to assume the identity of (impersonate) another authorized user.

- **SIGNAL statement [SP]**
Signals an exception condition.
- **START DATABASE statement**
Starts a database on the current database server.
- **START SERVER statement [Interactive SQL]**
Starts a database server.
- **START EXTERNAL ENVIRONMENT statement**
Starts an external environment.
- **START JAVA statement**
Starts the Java VM.
- **START LOGGING statement [Interactive SQL]**
Starts logging executed SQL statements and messages to a log file.
- **START SUBSCRIPTION statement [SQL Remote]**
Starts a subscription for a user to a publication.
- **START SYNCHRONIZATION DELETE statement [MobiLink]**
Restarts logging of deletes for MobiLink synchronization.
- **START SYNCHRONIZATION SCHEMA CHANGE statement [MobiLink]**
Starts a MobiLink synchronization schema change.
- **STOP DATABASE statement**
Stops a database on the current database server.
- **STOP EXTERNAL ENVIRONMENT statement**
Stops an external environment.
- **STOP JAVA statement**
Stops the Java VM.
- **STOP LOGGING statement [Interactive SQL]**
Stops logging of SQL statements and messages for the current session.
- **STOP SERVER statement**
Stops a database server.
- **STOP SUBSCRIPTION statement [SQL Remote]**
Stops a subscription for a user to a publication.
- **STOP SYNCHRONIZATION DELETE statement [MobiLink]**
Temporarily stops logging of deletes for MobiLink synchronization.
- **STOP SYNCHRONIZATION SCHEMA CHANGE statement [MobiLink]**
Stops a MobiLink synchronization schema change.
- **SYNCHRONIZE statement [MobiLink]**
Synchronizes a SQL Anywhere database with a MobiLink server. The synchronization options can be specified in the statement itself.
- **SYNCHRONIZE SUBSCRIPTION statement [SQL Remote]**
Synchronizes a subscription for a user to a publication.
- **SYSTEM statement [Interactive SQL]**
Launches an executable file from within Interactive SQL.
- **TRIGGER EVENT statement**
Triggers a named event. The event may be defined for event triggers or be a scheduled event.
- **TRUNCATE statement**

Deletes all rows from a table without deleting the table definition.

- **TRUNCATE TEXT INDEX statement**

Deletes the data in a MANUAL or an AUTO REFRESH text index.

- **TRY statement**

Implements error handling for compound statements (if an error occurs in the TRY block, it passes control to another group of statements that is enclosed in a CATCH block).

- **UNION statement**

Combines the results of two or more SELECT statements or query expressions.

- **UNLOAD statement**

Unloads data from a data source into a file.

- **UNPIVOT clause**

Unpivots compatible-type columns of a table expression in a FROM clause (FROM *unpivoted-derived-table expression*) into rows in a derived table. Unpivoting is used to normalize data, for example when you have similar data stored in multiple columns in tables and you want to return it in one column.

- **UPDATE (positioned) statement [ESQL] [SP]**

Modifies the data at the current location of a cursor.

- **UPDATE statement [SQL Remote]**

Modifies data in the database.

- **UPDATE statement**

Modifies rows in database tables.

- **VALIDATE LDAP SERVER statement**

Validates an LDAP server configuration object.

- **VALIDATE statement**

Validates the current database, one or more tables, materialized views, or indexes in the current database.

- **WAITFOR statement**

Delays processing for the current connection for a specified amount of time or until a given time.

- **WAITFOR SEMAPHORE statement**

Decrements the counter associated with a semaphore.

- **WHENEVER statement [ESQL]**

Specifies error handling in Embedded SQL programs.

- **WHILE statement [T-SQL]**

Provides repeated execution of a statement or compound statement.

- **WINDOW clause**

Defines all or part of a window for use with window functions such as AVG and RANK in a SELECT statement.

- **WRITETEXT statement [T-SQL]**

Permits non-logged updating of a CHAR, NCHAR, or BINARY column. This feature is provided solely for compatibility with Transact-SQL and its use is not recommended.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

ALLOCATE DESCRIPTOR statement [ESQL]

Allocates space for a SQL descriptor area (SQLDA).

Syntax

```
ALLOCATE DESCRIPTOR descriptor-name  
[ WITH MAX { integer | hostvar } ]
```

descriptor-name : *identifier*

Parameters

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.

Remarks

Allocates space for a descriptor area (SQLDA). You must declare the following in your C code before using this statement:

```
struct sqlda * descriptor_name
```

Privileges

None.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** `ALLOCATE DESCRIPTOR` is part of optional ANSI/ISO SQL Language Feature B031 "Basic dynamic SQL".

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 *      sqllda1;
    if( !db_init( &sqlca ) ) {
        return 1;
    }
    db_string_connect( &sqlca,
        "UID=DBA;PWD=passwd;DBF=d:\\DB Files\\sample.db");
    EXEC SQL ALLOCATE DESCRIPTOR sqllda1 WITH MAX 25;
    EXEC SQL PREPARE :stmt FROM
        'SELECT * FROM Employees';
    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;
        EXEC SQL DESCRIBE :stmt into sqllda1;
    }
    type = DT_STRING; // change the type to string
    EXEC SQL SET DESCRIPTOR sqllda1 VALUE 2 TYPE = :type;
    fill_sqllda( sqllda1 );
    // 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;
}
```

Related Information

[The SQL descriptor area \(SQLDA\)](#)

The SQLDA (SQL Descriptor Area) is an interface structure that is used for dynamic SQL statements. The structure is used to pass information regarding host variables and SELECT statement results to and from the database. The SQLDA is defined in the header file *sqlda.h*.

[DEALLOCATE DESCRIPTOR statement \[ESQL\]](#)

Frees memory associated with a SQL descriptor area.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

ALTER DATABASE statement

Upgrades the database, turns jConnect support for a database on or off, calibrates the database, changes the transaction log and transaction log mirror file names, or forces a mirror server to take ownership of a database.

Syntax

- **Performing a system upgrade, or restoring objects**

```
ALTER DATABASE UPGRADE
[ PROCEDURE ON ]
[ JCONNECT { ON | OFF } ]
[ RESTART { ON | OFF } ]
[ SYSTEM PROCEDURE AS DEFINER { ON | OFF } ]
```

- **Performing a user-defined upgrade**

```
ALTER DATABASE UPGRADE
SCRIPT FILE sql_script_path
[ RESTART ON | OFF ]
```

- **Performing calibration**

```
ALTER DATABASE {
  CALIBRATE [ SERVER ]
  | CALIBRATE DBSPACE dbspace-name
  | CALIBRATE DBSPACE TEMPORARY
  | CALIBRATE GROUP READ
  | CALIBRATE PARALLEL READ
  | RESTORE DEFAULT CALIBRATION
}
```

- **Changing transaction log and transaction log mirror names**

```
ALTER DATABASE dbfile
ALTER [ TRANSACTION ] LOG
{ ON [ log-name ] [ MIRROR mirror-name ] | OFF }
[ KEY key ]
```

- **Changing ownership of a database**

```
ALTER DATABASE
{ dbname FORCE START
  | SET PARTNER FAILOVER }
```

- **Turning off checksum**

```
ALTER DATABASE SET CHECKSUM OFF
```

- **Cache warming to a steady state**

```

ALTER DATABASE
{ SAVE CACHE
| RESTORE CACHE
| DROP CACHE }

```

Parameters

dbfile The database file. You can also specify a variable name.

PROCEDURE clause Drop and recreate all dbo- and SYS-owned procedures in the database.

JCONNECT clause To allow the jConnect JDBC driver access to system catalog information, specify JCONNECT ON. This clause installs the system objects that provide jConnect support. Specify JCONNECT OFF to exclude the jConnect system objects. You can still use JDBC, as long as you do not access system information. JCONNECT is ON by default.

RESTART clause RESTART is ON by default. When RESTART ON is specified and the AutoStop connection parameter is set to No, the database restarts after it is upgraded. Otherwise, the database is stopped after an upgrade.

SYSTEM PROCEDURE AS DEFINER { ON | OFF } clause The SYSTEM PROCEDURE AS DEFINER clause specifies whether to execute pre-16.0 system procedures that perform privileged tasks with the privileges of the invoker or the definer (owner). ON means these system procedures are executed with the privileges of the definer (owner). OFF means these system procedures are executed with the privileges of the invoker.

If this clause is not specified, the default is to maintain the current behavior of the database being upgraded. When upgrading a pre-16.0 database, this means running the procedures as definer.

This setting does not impact user-defined procedures, or any procedures introduced in version 16.0 or later.

SCRIPT FILE clause Use this clause to specify the location of the user-defined upgrade script file. *sql_script_path* is the location and name of a .sql script file containing the DML and DDL statements to perform.

If the script execution is not successful, and RESTART ON is specified (the default), the database is rolled back to the checkpoint that was automatically performed before the upgrade, and restarted. If RESTART OFF is specified, the database is stopped without a checkpoint, and is rolled back to the last checkpoint the next time the database is started.

CALIBRATE [SERVER] clause Calibrate all dbspaces except for the temporary dbspace. This clause also performs the work done by CALIBRATE PARALLEL READ.

CALIBRATE DBSPACE clause Calibrate the specified dbspace.

CALIBRATE DBSPACE TEMPORARY clause Calibrate the temporary dbspace.

CALIBRATE GROUP READ clause Perform group read calibration on the temporary dbspace. Writes large work tables to the temporary dbspace and uses different group read sizes to time the reading of the files. If adding space to the temporary table exceeds the limit for the connection, or if the cache is not large enough to allow calibration with the largest memory size, calibration fails and an error message is returned.

CALIBRATE PARALLEL READ clause Calibrate the parallel I/O capabilities of devices for all dbspace files. The CALIBRATE [SERVER] clause also performs this calibration.

RESTORE DEFAULT CALIBRATION clause Restore the Disk Transfer Time (DTT) model

to the built-in default values that are based on typical hardware and configuration settings.

ALTER [TRANSACTION] LOG clause Change the file name of the transaction log or transaction log mirror file. If MIRROR *mirror-name* is not specified, the clause sets a file name 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 starts using the new file as its transaction log.

If MIRROR *mirror-name* is specified, the clause sets a file name 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 starts using the new file as its transaction log mirror.

You can also use this clause to turn off the transaction log or transaction log mirror. For example, ALTER DATABASE ALTER LOG OFF.

KEY clause Specify the encryption key to use for the transaction log or transaction log mirror. The encryption key can be either a string or a variable name. When using the ALTER [TRANSACTION] LOG clause on a strongly encrypted database, you must specify the encryption key.

dbname FORCE START clause Force a database server that is currently acting as the mirror server to take ownership of the database.

Caution

Using the FORCE START clause can result in the loss of transactions if the primary server contains transactions that the mirror server does not have.

It is recommended that you restart the primary and execute ALTER DATABASE with the SET PARTNER FAILOVER clause to force a failure without lost transactions. The FORCE START clause should only be used as a last resort when the primary cannot be restarted.

This clause can be executed from within a procedure or event, and must be executed while connected to the utility database on the mirror server.

SET PARTNER FAILOVER clause Initiate a database mirroring failover from the primary server to the mirror server without stopping the server. This statement must be executed while you are connected to the database on the primary server, and can be executed from within a procedure or event. When this statement is executed:

1. The database server closes all connections to the database, including the connection that executed the statement
2. The database stops, and then restarts in the mirror role.
3. If the statement is contained in a procedure or event, then the other statements that follow it may not be executed

SET CHECKSUM OFF clause Disable global checksums for the database. By default, new databases have global checksums enabled, while version 11 and earlier databases do not have global checksums enabled.

Regardless of the setting of this clause, the database server always enables write checksums for databases running on storage devices such as removable drives, to help provide early detection if the database file becomes corrupt. The database server also calculates checksums for critical pages during validation activities.

For databases that do not have global checksums enabled, you can enable write checksums by

using the -wc options.

SAVE CACHE clause Record the current cache contents so that you can restore the database to this state when necessary to improve performance.

This statement is not logged to the transaction log and cannot be executed on read-only databases, including databases involved in high availability and read-only scale-out configurations.

After the statement executes, the prefetch_pages column of the SYSDBSPACE system view is updated with one bit in the page for each page currently in cache. The bitmap that is used is the same one as that returned from the sp_db_cache_contents system procedure.

RESTORE CACHE clause Improve performance by restoring the current database to the state it was in when the ALTER DATABASE SAVE CACHE statement was executed.

This statement is not logged to the transaction log.

This statement reads the pages identified by the prefetch_pages column of the SYSDBSPACE system view. The effect of the statement is the same as that of calling the sp_read_db_pages system procedure for each dbspace with the bitmap stored in the prefetch_column of the ISYSDBSPACE system table.

DROP CACHE clause Clear any saved cache pages.

This clause clears the pages stored in the saved_cache_pages column of the SYSDBSPACE system view and sets the column value to NULL. After executing the ALTER DATABASE DROP CACHE statement, subsequent ALTER DATABASE RESTORE CACHE statements return an error unless the ALTER DATABASE SAVE CACHE statement is executed first.

Remarks

- **Syntax - Upgrading components or restoring objects** Use the ALTER DATABASE UPGRADE statement as an alternative to the Upgrade utility (dbupgrad) to upgrade or update a database. By default, the database is stopped and restarted after the upgrade. The transaction log is archived during the upgrade and a new transaction log is created before the database is stopped or restarted.

After executing an ALTER DATABASE UPGRADE statement, you should shut down the database and archive the transaction log.

In general, changes in databases between minor versions are limited to additional database options and minor system table and system procedure changes. The ALTER DATABASE UPGRADE statement upgrades the system tables to the current version and adds any new database options. If necessary, it also drops and recreates all system procedures. You can force a rebuild of the system procedures by specifying the PROCEDURE ON clause.

An error message is returned if you execute an ALTER DATABASE UPGRADE statement on a database that is currently being mirrored.

You can also use the ALTER DATABASE UPGRADE statement to restore settings and system objects to their original installed state.

Features that require a physical reorganization of the database file are not made available by executing an ALTER DATABASE UPGRADE statement. Such features include index enhancements and changes in data storage. To obtain the benefits of these enhancements, you must unload and reload your database.

Back up your database files before attempting to upgrade your database.

To use the jConnect JDBC driver to access system catalog information, specify JCONNECT ON (the default). To exclude the jConnect system objects, specify JCONNECT OFF. Setting JCONNECT OFF does not remove jConnect support from a database. You can still use JDBC, as long as you do not access system catalog information. If you subsequently download a more recent version of jConnect, you can upgrade the version in the database by (re-)executing the ALTER DATABASE UPGRADE JCONNECT ON statement.

- **Syntax - Performing a user-defined upgrade** If *sql_script_path* is not valid, an error is returned and the database is not stopped or restarted.

The script must be stored on, and run from, the database server computer.

If the script execution is successful, the database continues running.

- **Syntax - Performing calibration** You can perform recalibration of the I/O cost model used by the optimizer. This operation updates the Disk Transfer Time (DTT) model, which is a mathematical model of the disk I/O used by the cost model. When you recalibrate the I/O cost model, the database server is unavailable for other use. In addition, it is essential that all other activities on the computer are idle. Recalibrating the database server is an expensive operation and may take some time to complete. Leave the default in place.

When using the CALIBRATE PARALLEL READ clause, parallel calibration is not performed on dbspace files with fewer than 10000 pages. Even though the database server automatically suspends all of its activity during calibration operations, parallel calibration should be done when there are no processes consuming significant resources on the same computer. After calibration, you can retrieve the maximum estimated number of parallel I/O operations allowed on a dbspace file by using the IOParallelism extended database property.

To eliminate repetitive, time-consuming recalibration activities when there is a large number of similar hardware installations, you can re-use a calibration by unloading it and then applying it (loading it) into another database by using the *sa_unload_cost_model* and *sa_load_cost_model* system procedures, respectively.

- **Syntax - Changing transaction log and transaction log mirror names** Use the ALTER DATABASE statement to change the transaction log and transaction log mirror names associated with a database file. The database must not be running to make these changes. These changes are the same as those made by the Transaction Log (dblog) utility. You can execute this statement while connected to the utility database or another database, depending on the setting of the -gu option.

If you are changing the transaction log or transaction log mirror of an encrypted database, you must specify a key. You cannot stop using the transaction log if the database is using auditing. Once you turn off auditing, you can stop using the transaction log. This syntax is not supported in procedures, triggers, events, or batches.

Use the BACKUP DATABASE statement to rename the transaction log for a running database. For example:

```
BACKUP DATABASE DIRECTORY 'directory-name'  
TRANSACTION LOG ONLY  
TRANSACTION LOG RENAME;
```

- **Syntax - Changing ownership of a database** ALTER DATABASE...FORCE START must be executed from the mirror server, not the primary server.
- **Syntax - Turning off checksum** This clause can only be used to disable checksums for a

database.

- **Syntax - Cache warming to a steady state** Use the ALTER DATABASE statement to record the cache contents at a steady state and restore this state when necessary.

Note For parameters that accept variable names, an error is returned if one of the following conditions is true:

- The variable does not exist
- The contents of the variable are NULL
- The variable exceeds the length allowed by the parameter
- The data type of the variable does not match that required by the parameter

Privileges

Unless otherwise specified in the following list, only the ALTER DATABASE system privilege is required to upgrade the database.

- **Upgrading components or restoring objects:** you must have the ALTER DATABASE system privilege, and must be the only connection to the database.
- **Performing a user-defined upgrade:** While only the ALTER DATABASE system privilege is required to execute the ALTER DATABASE statement, the user performing the upgrade must have all the privileges required to perform the actions in the specified .sql file. If the user does not have the proper privileges, the database upgrade fails and the database is rolled back to its state prior to executing the ALTER DATABASE statement.
- **Performing calibration:** you must have the SERVER OPERATOR system privilege, you must have file permissions on the directories where the transaction log is located, and the database must not be running.

Your ability to execute the ALTER DATABASE *dbfile* ALTER TRANSACTION LOG statement depends on the setting for the -gu database option, and whether you have the SERVER OPERATOR system privilege.

- **Changing ownership of a database:** you must have the SERVER OPERATOR system privilege.

The privileges required to execute an ALTER DATABASE dbname FORCE START statement can be changed by the -gd database server option.

- **Turning off checksum:** you must have the ALTER DATABASE system privilege.
- **Cache warming to a steady state:** you must have the SERVER OPERATOR system privilege.

Side effects

Automatic commit

When executing an ALTER DATABASE UPGRADE statement, the transaction log is archived during the upgrade, and a new transaction log is created when the database restarts after the upgrade. Also, by default the database is stopped and restarted after the upgrade.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.
- **Transact-SQL** The ALTER DATABASE statement is supported by Adaptive Server Enterprise. However, the statement's clauses supported by Adaptive Server Enterprise are disjoint from

those clauses supported by SQL Anywhere.

Example

1. The following statement disables jConnect support:

```
ALTER DATABASE UPGRADE JCONNECT OFF;
```

2. The following statement sets the transaction log file name associated with *demo.db* to *mynewdemo.log*:

```
ALTER DATABASE 'demo.db\\demo.db'  
  ALTER LOG ON 'mynewdemo.log';
```

3. The following statement performs a user-defined upgrade by running a fictitious *myUpgrade.sql* script file. If the upgrade is successful, the database continues to run. If the upgrade is not successful, the database is rolled back to the checkpoint that was automatically performed before the upgrade, and restarted:

```
ALTER DATABASE UPGRADE SCRIPT FILE 'C:\\Users\\Public\\Documents  
\\myUpgrade.sql'  
  RESTART ON;
```

4. The following statement creates a variable for the database *asatest.db*:

```
CREATE VARIABLE @db1 LONG VARCHAR ;  
SET @db1 = 'asatest' ;
```

- a. The following statement uses the variable @db1 to alter the database *asatest.db*:

```
ALTER DATABASE @db1  
  ALTER TRANSACTION LOG ON 'vis_tmp2.log'  
  MIRROR 'vis_tmp2.mlg' ;
```

5. The following example shows you how to save the contents of the cache when the database is running in a steady state and then restore the cache to that state when necessary.

- a. Create a dbspace by executing the following statement:

```
CREATE DBSPACE mydbs  
  AS 'C:\\mydb\\mydbs.db';
```

- b. Create a table in the new dbspace and add some data by executing the following statements:

```
CREATE TABLE mytable( col1 INT, col2 CHAR(128) ) IN mydbs;  
  
INSERT INTO mytable  
  SELECT column_id, column_name  
  FROM sys.syscolumn;
```

```
COMMIT;
```

- c. Execute the following statement to determine the ID of the new dbspace:

```
SELECT * FROM SYS.SYSDBSpace WHERE dbspace_name = 'mydbs';
```

Note the ID of the new dbspace and that the saved_cache_pages column is NULL.

- d. Verify which pages from the new dbspace are currently in the cache by executing the following statement:

```
SELECT * FROM dbo.sp_db_cache_contents( );
```

The statement above returns information for all dbspaces in the database. To restrict the results to the new mydbs dbspace, execute the following statement, where *mydbs-ID* is the ID of the new mydbs dbspace that was obtained from SYS. SYSDBSpace:

```
SELECT * FROM dbo.sp_db_cache_contents( mydbs-ID );
```

- e. Save the current steady state of the database by executing the following statement:

```
ALTER DATABASE SAVE CACHE;
```

- f. Later, after numerous other transactions have been executed and the cache has undergone significant change, you want to return to the saved steady state. You can either read all the steady state pages for all dbspaces into the cache or read all the steady state pages for a specific dbspace into the cache.

To read all the steady state pages for all dbspaces into the cache, execute the following statement:

```
ALTER DATABASE RESTORE CACHE;
```

If you would rather read all the steady state pages for one dbspace (for example, the new dbspace mydbs), then execute a series of statements similar to the following:

```
CREATE VARIABLE cache_pages LONG VARBIT;
```

```
SELECT saved_cache_pages  
      INTO cache_pages  
      FROM SYS.SYSDBSpace  
      WHERE dbspace_name='mydbs';
```

```
CALL dbo.sp_read_db_pages( mydbs-ID, cache_pages );
```

Related Information

[Running pre-16.0 system procedures as invoker or definer](#)

Some system procedures present in the software before version 16.0 that perform privileged tasks in

the database, such as altering tables, can be run with either the privileges of the invoker, or of the definer (owner).

[User-defined upgrades](#)

In addition to performing system upgrades, you can define your own upgrade process by storing the SQL statements for a user-defined upgrade in a `.sql` script file.

[Database rebuilds](#)

Rebuilding a database is a specific type of import and export involving unloading and reloading your database.

[Upgrades and rebuilds in a database mirroring system](#)

All database servers in a database mirroring system must use the same minor release.

[Troubleshooting: The primary server cannot be restarted](#)

Force the mirror server to take over as the primary server in situations where all other attempts to restart have failed.

[CREATE DATABASE statement](#)

Creates a database.

[CREATE STATISTICS statement](#)

Recreates the column statistics used by the optimizer, and stores them in the ISYSCOLSTAT system table.

[BACKUP DATABASE statement](#)

Backs up a database and transaction log.

[Upgrade utility \(dbupgrad\)](#)

Updates the system tables and views, adds new database options, recreates all system stored procedures, installs jConnect support, changes support for Java in the database, archives the transaction log, and creates a new transaction log.

[DB_EXTENDED_PROPERTY function \[System\]](#)

Returns the value of the given property. Allows an optional property-specific string parameter to be specified.

[-gk database server option](#)

Sets the privileges required to stop the database server.

[-gu database server option](#)

Sets the privilege required for executing database file administration statements such as for creating or dropping databases.

[-wc database server option](#)

Controls whether checksums are enabled on write operations for all databases on this database server, if the databases do not have checksums enabled by default.

[-wc database option](#)

Controls whether checksums are enabled on write operations for the database, if it does not have checksums enabled by default.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

ALTER DBSPACE statement

Preallocates space for a dbspace or for the transaction log, or updates the catalog when a dbspace file is renamed or moved.

Syntax

```
ALTER DBSPACE { dbspace-name | TRANSLOG | TEMPORARY }  
  { ADD number [ add-unit ]  
    | RENAME filename }
```

add-unit :

PAGES

| **KB**

| **MB**

| **GB**

| **TB**

Parameters

TRANSLOG clause You supply the special dbspace name TRANSLOG to preallocate disk space for 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 many binary large objects (BLOBs) such as bitmaps.

The syntax ALTER DBSPACE *dbspace-name* TRANSLOG RENAME *filename* is not supported.

TEMPORARY clause You supply the special dbspace name TEMPORARY to add space to temporary dbspaces. When space is added to a temporary dbspace, the additional space materializes in the corresponding temporary file immediately. Preallocating space to the temporary dbspace of a database can improve performance during execution complex queries that use large work tables.

ADD clause An ALTER DBSPACE statement with the ADD clause preallocates disk space for a dbspace. It extends the corresponding database file by the specified size, in units of pages, kilobytes (KB), megabytes (MB), gigabytes (GB), or terabytes (TB). If you do not specify a unit, PAGES is the default. The page size of a database is fixed when the database is created.

If space is not preallocated, database files are extended by about 256 KB at a time for page sizes of 2 KB, 4 KB, and 8 KB, and by about 32 pages for other page sizes, when the space is needed. Pre-allocating space can improve performance for loading large amounts of data and also serves to keep the database files more contiguous within the file system.

You can use this clause to add space to any of the predefined dbspaces (system, temporary, temp, translog, and translogmirror).

RENAME clause If you rename or move a database file other than the main file to a different directory or device, you can use ALTER DBSPACE with the RENAME clause to ensure that the database server finds the new file when the database is started. The *filename* parameter can be a string literal, or a variable.

The name change takes effect as follows:

- If the dbspace was already open before the statement was executed (that is, you have not yet renamed the actual file), it remains accessible; however, the name stored in the catalog

is updated. After the database is stopped, you must rename the file to match what you provided using the RENAME clause, otherwise the file name won't match the dbspace name in the catalog and the database server is unable to open the dbspace the next time the database is started.

- If the dbspace was not open when the statement was executed, the database server attempts to open it after updating the catalog. If the dbspace can be opened, it becomes accessible. No error is returned if the dbspace cannot be opened.

To determine if a dbspace is open, execute the statement below. If the result is NULL, the dbspace is not open.

```
SELECT DB_EXTENDED_PROPERTY('FileSize','dbspace-name');
```

Using ALTER DBSPACE with RENAME on the main dbspace, system, has no effect. The RENAME clause is not supported for changing the name of the transaction log file. You can use the BACKUP DATABASE statement to rename the transaction log for a running database. For example:

```
BACKUP DATABASE DIRECTORY 'directory-name'  
TRANSACTION LOG ONLY  
TRANSACTION LOG RENAME;
```

Remarks

Each database is held in one or more files. A dbspace is an additional file with a logical name associated with each database file, and used to hold more data than can be held in the main database file alone. ALTER DBSPACE modifies the main dbspace (also called the root file) or an additional dbspace. The dbspace names for a database are held in the SYSDBSpace system view. The main database file has a dbspace name of system.

When a multi-file database is started, the start line or ODBC data source description tells the database server where to find the main database file. The main database file holds the system tables. The database server looks in these system tables to find the location of the other dbspaces, and then opens each of the other dbspaces. You can specify which dbspace new tables are created in by setting the default_dspace option.

Privileges

You must have the MANAGE ANY DBSPACE system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example increases the size of the system dbspace by 200 pages:

```
ALTER DBSPACE system  
ADD 200;
```

The following example increases the size of the system dbspace by 400 MB:

```
ALTER DBSPACE system  
ADD 400 MB;
```

The following example changes the file name associated with the fictitious system_2 dbspace:

```
ALTER DBSPACE system_2  
RENAME 'e:\db\dbspace2.db';
```

Related Information

[Predefined dbspaces](#)

The database server uses predefined dbspaces for its databases.

[Database file types](#)

Each database has several files associated with it.

[CREATE DBSPACE statement](#)

Defines a new database space and creates the associated database file.

[BACKUP DATABASE statement](#)

Backs up a database and transaction log.

[SYSDBSpace system view](#)

Each row in the SYSDBSpace system view describes a dbspace file. The underlying system table for this view is ISYDBSPACE.

[default_dbspace option](#)

Changes the default dbspace in which tables are created.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

ALTER DOMAIN statement

Renames a user-defined domain or data type.

Syntax

```
ALTER { DOMAIN | DATATYPE } user-type  
RENAME new-name
```

Remarks

When you execute this statement, the name of the user-defined domain or data type is updated in the ISYSUSERTYPE system table.

Note Any procedures, triggers, views, or events that refer to the user-defined domain or data type must be recreated, or else they continue to refer to the old name.

Privileges

You must be the owner of the domain, or have one of the following privileges:

- ALTER privilege on the domain
- ALTER DATATYPE system privilege
- ALTER ANY OBJECT system privilege

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard. The ALTER DOMAIN statement is optional ANSI/ISO SQL Feature F711. However, in the standard, ALTER DOMAIN can specify modified DEFAULT or CHECK constraint clauses for an existing domain. Neither of these operations are supported in the software. Feature F711 does not support the renaming of a domain.

Example

The following example renames the fictitious Address domain to MailingAddress:

```
ALTER DOMAIN Address RENAME MailingAddress;
```

Related Information

[Domains](#)

Domains are aliases for built-in data types, including precision and scale values where applicable, and optionally including DEFAULT values and CHECK conditions. Some domains, such as the monetary data types, are predefined, but you can add more of your own.

[How to use domains to improve data integrity](#)

A **domain** is a user-defined data type that can restrict the range of acceptable values or provide defaults.

[SYSUSERTYPE system view](#)

Each row in the SYSUSERTYPE system view holds a description of a user-defined data type. The underlying system table for this view is ISYSUSERTYPE.

[CREATE DOMAIN statement](#)

Creates a domain in a database.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)» [Alphabetical list of SQL statements](#)

ALTER EVENT statement

Changes the definition of an event or its associated handler for automating predefined actions, or alters the definition of scheduled actions. You can also use this statement to hide the definition of an event handler.

Syntax

- Altering an event

```
ALTER EVENT event-name
[ AT { CONSOLIDATED | REMOTE | ALL } ]
[ FOR { PRIMARY | ALL } ]
[ { DELETE TYPE
  | TYPE event-type
  | WHERE { trigger-condition | NULL }
  | { ADD | ALTER | DELETE } SCHEDULE schedule-spec } ]
[ ENABLE | DISABLE ]
[ [ ALTER ] HANDLER compound-statement | DELETE HANDLER ]
```

event-type :

```
BackupEnd
| Connect
| ConnectFailed
| DatabaseStart
| DBDiskSpace
| Deadlock
| "Disconnect"
| GlobalAutoincrement
| GrowDB
| GrowLog
| GrowTemp
| LogDiskSpace
| RAISERROR
| ServerIdle
| TempDiskSpace
```

trigger-condition :

```
event_condition( condition-name ) { = | < | > | != | <= | >= } value |
@variable-name
```

schedule-spec :

```
[ schedule-name ]
{ START TIME start-time | BETWEEN start-time AND end-time }
[ EVERY period { HOURS | MINUTES | SECONDS } ]
[ ON { ( day-of-week, ... ) | ( day-of-month, ... ) } ]
[ START DATE start-date ]
```

event-name | *schedule-name* : *identifier*

day-of-week : *string*

value | *period* | *day-of-month* : *integer*

start-time | *end-time* : *time*

start-date : *date*

- **Hiding the definition of an event handler**

ALTER EVENT *event-name* **SET HIDDEN**

Parameters

ALTER EVENT clause Events do not have owners. If you specify an owner (for example, *owner.event-name*) the owner portion is ignored.

AT clause Use this clause to change the specification regarding the databases at which the event is handled.

FOR clause Use this clause in a database mirroring or read-only scale-out system to restrict the databases at which the event is handled.

DELETE TYPE clause Use this clause to remove an association of the event with an event type.

ADD | ALTER | DELETE SCHEDULE clause Use this clause to change the definition of a schedule. Only one schedule can be altered in any one ALTER EVENT statement.

WHERE clause Use this clause to change the trigger condition under which an event is fired. The WHERE NULL option deletes a condition.

You can specify a variable name for the *event_condition* value.

START TIME clause Use this clause to specify the start time and, optionally, the end time, for the event. The *start-time* and *end-time* parameters are strings (for example, '12:34:56'). Expressions are not allowed (for example, *NOW()*).

You can specify a variable name for *start-time*. If *start-time* is a NULL string variable, it is ignored.

BETWEEN...AND clause You can specify a variable name for *start-time* and *end-time*. If *start-time* or *end-time* are NULL string variables, they are ignored.

EVERY clause You can specify a variable name for *period*. If *period* is a NULL integer variable, the EVERY clause is ignored.

START DATE clause Use this clause to specify the start date for the event. The *start-date* parameter is a string. Expressions are not allowed (for example, *TODAY()*). If *start-date* is a NULL string variable, it is ignored.

You can specify a variable name for *start-date*.

SET HIDDEN clause Use this clause to hide the definition of an event handler. Specifying the SET HIDDEN clause results in the permanent obfuscation of the event handler definition stored in the action column of the ISYSEVENT system table.

Remarks

This statement allows you to alter an event definition created with CREATE EVENT. Possible uses include the following:

- hiding the definition of an event handler
- defining and testing an event handler without a trigger condition or schedule during a development phase, and then adding the conditions for execution using ALTER EVENT once the event handler is completed

Events are not owned. However, when an event is created, a user name is associated with the event (either by explicitly specifying a user name in the CREATE EVENT statement, or implicitly as the creator). The event runs with the privileges of that user name. You cannot use the ALTER EVENT statement to change the user name associated with an event (if you specify a user name, it is ignored). Instead, you must drop and recreate the event, and specify the new user name.

If you need to alter an event, you can disable it while it is running by executing an ALTER EVENT...DISABLE statement. To disable an event in SQL Central, right-click the event and clear the **Enabled** option. Disabling the event does not interrupt current event handler execution; the event handler continues to execute until completion. When the event handler completes, it is not restarted until you re-enable it. You can alter and then re-enable the definition. To determine what events are running, execute the following statement:

```
SELECT *  
FROM dbo.sa_conn_info()  
WHERE CONNECTION_PROPERTY( 'EventName', Number ) = 'event-name';
```

An *owner* specification before *event-name* is ignored.

Note For *required* parameters that accept variable names, an error is returned if one of the following conditions is true:

- The variable does not exist
- The contents of the variable are NULL
- The variable exceeds the length allowed by the parameter
- The data type of the variable does not match that required by the parameter

Privileges

You must have either the ALTER ANY OBJECT or MANAGE ANY EVENT system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

1. The following example creates then alters an event using variables for the BETWEEN clause.
 - a. The following statements create three variables, one for start time, one for end time, and one for interval time:

```

CREATE VARIABLE @st1 LONG VARCHAR
CREATE VARIABLE @et1 LONG VARCHAR
CREATE VARIABLE @int1 INTEGER
SET @st1 = ' 8:00AM '
SET @et1 = ' 6:00PM '
SET @int1 = 1;

```

- b. The following statement creates an event that backs up the database's transaction log and uses the created variables for the BETWEEN clause:

```

CREATE EVENT HourlyLogBackup
  SCHEDULE hourly_log_backup
  BETWEEN @st1 AND @et1
  EVERY @int1 HOURS ON
    ( 'Monday' , 'Tuesday' , 'Wednesday' , 'Thursday' , 'Friday' )
  HANDLER
    BEGIN
      BACKUP DATABASE DIRECTORY 'C:\\database\\backup'
      TRANSACTION LOG ONLY
      TRANSACTION LOG RENAME
    END;

```

- c. The following statement resets the variable @st1 to 'Now':

```

SET @st1 = ' Now ';

```

- d. The following statement alters the event by changing the start time to 'Now':

```

ALTER EVENT HourlyLogBackup
  SCHEDULE hourly_log_backup
  BETWEEN @st1 AND @et1
  EVERY @int1 HOURS ON
    ( 'Monday' , 'Tuesday' , 'Wednesday' , 'Thursday' , 'Friday' )
  HANDLER
    BEGIN
      BACKUP DATABASE DIRECTORY 'C:\\database\\backup'
      TRANSACTION LOG ONLY
      TRANSACTION LOG RENAME
    END;

```

2. The following example creates an event that runs incremental backups and then alters the event using variables for the START TIME, EVERY, and START DATE clauses.

- a. The following statement creates an event that runs an incremental backup daily at 1 AM.

```
CREATE EVENT IncrementalBackup
SCHEDULE
  START TIME '1:00 AM' EVERY 24 HOURS
  HANDLER
  BEGIN
    BACKUP DATABASE DIRECTORY 'c:\\backup'
    TRANSACTION LOG ONLY
    TRANSACTION LOG RENAME MATCH
  END;
```

3. The following statements create three variables, one for start time, one for start date, and one for interval time:

```
CREATE VARIABLE @st2 LONG VARCHAR
CREATE VARIABLE @sd2 LONG VARCHAR
CREATE VARIABLE @int2 INTEGER
SET @st2 = ' 3:00AM '
SET @sd2 = '2013-01-01'
SET @int2 = 24;
```

4. The following statement alters the event IncrementalBackup and uses variables in the START TIME, EVERY, and START DATE clauses:

```
ALTER EVENT IncrementalBackup
SCHEDULE
  START TIME @st2 EVERY @int2 HOURS
  START DATE @sd2
  HANDLER
  BEGIN
    BACKUP DATABASE DIRECTORY 'c:\\backup'
    TRANSACTION LOG ONLY
    TRANSACTION LOG RENAME MATCH
  END;
```

Related Information

[System events](#)

Each system event provides a hook on which you can program a set of actions.

[Hiding an event handler](#)

Hide the definition for an event handler using the ALTER EVENT statement.

[SYSEVENT system view](#)

Each row in the SYSEVENT system view describes an event created with CREATE EVENT. The underlying system table for this view is ISYSEVENT.

[BEGIN statement](#)

Specifies a compound statement.

[CREATE EVENT statement](#)

Defines an event and its associated handler for automating predefined actions, and to define scheduled actions.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

ALTER EXTERNAL ENVIRONMENT statement

Specifies the location of an external environment such as Java, PHP, or Perl.

Syntax

```
ALTER EXTERNAL ENVIRONMENT environment-name  
LOCATION location-string
```

environment-name :

```
C_ESQL32  
| C_ESQL64  
| C_ODBC32  
| C_ODBC64  
| CLR  
| DBMSYNC  
| JAVA  
| JS  
| PERL  
| PHP
```

Parameters

environment-name Use *environment-name* to specify the external environment you are altering.

location-string Use *location-string* to specify the location on the database server computer where the executable/binary for the external environment can be found. It includes the executable/binary name. This path can either be fully qualified or relative. If the path is relative, then the executable/binary must be in a location where the server can find it.

Remarks

In general, the ALTER EXTERNAL ENVIRONMENT statement is used to help the database server locate an external environment module when it cannot be located using standard search techniques (such as searching the software install location, system PATH, etc.).

CLR The LOCATION string is used to identify both the location and version of the CLR external environment support module. For example, *dbextclr[VER_MAJOR]_v4.0* indicates that support for .NET 4.0 is required. In this example, the file path is not specified so the database server will locate the module using standard search techniques. The *dbextclr[VER_MAJOR]* module supports .NET 2 and 3.5. The *dbextclr[VER_MAJOR]_v4.5* module supports .NET 4.5. For portability to newer versions of the software, use [VER_MAJOR] in the LOCATION string rather than the current software release version number. The database server will replace [VER_MAJOR] with the appropriate version number.

DBMSYNC In cases where there are multiple versions of the database server software installed on the same system, or if the dbmsync executable is not located in the same directory as the database server and cannot be found in the PATH environment variable, use the ALTER EXTERNAL ENVIRONMENT command to specify the location of the dbmsync executable.

Privileges

You must have the **MANAGE ANY EXTERNAL ENVIRONMENT** system privilege.

Side effects

None

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example specifies the location of the Perl executable for use when using Perl as an external environment.

```
ALTER EXTERNAL ENVIRONMENT PERL  
LOCATION 'c:\\Perl64\\bin\\perl.exe';
```

Related Information

[External environment support](#)

Six external runtime environments are supported. These include Embedded SQL and ODBC applications written in C/C++, and applications written in Java, JavaScript, Perl, PHP, or languages such as C# and Visual Basic that are based on the Microsoft .NET Framework Common Language Runtime (CLR).

[START EXTERNAL ENVIRONMENT statement](#)

Starts an external environment.

[STOP EXTERNAL ENVIRONMENT statement](#)

Stops an external environment.

[INSTALL EXTERNAL OBJECT statement](#)

Installs an object that can be run in an external environment.

[REMOVE EXTERNAL OBJECT statement](#)

Removes an external object from the database.

[SYSEXTERNENV system view](#)

Each row in the SYSEXTERNENV system view describes the information needed to identify and launch each of the external environments. The underlying system table for this view is ISYSEXTERNENV.

[SYSEXTERNENVOBJECT system view](#)

Each row in the SYSEXTERNENVOBJECT system view describes an installed external object. The underlying system table for this view is ISYSEXTERNENVOBJECT.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

ALTER FUNCTION statement

Modifies a function.

Syntax

- **Change the definition of a function**

ALTER FUNCTION [*owner.*] *function-name* *function-definition*

function-definition : [see the CREATE FUNCTION statement](#)

- **Obfuscate a function definition**

ALTER FUNCTION [*owner.*] *function-name*
SET HIDDEN

- **Recompile a function**

ALTER FUNCTION [*owner.*] *function-name*
RECOMPILE

Remarks

You must include the entire new function in the ALTER FUNCTION statement.

- **Change the definition of a function** The ALTER FUNCTION statement is identical in syntax to the CREATE FUNCTION statement except for the first word.

With ALTER FUNCTION, existing privileges on the function remain unmodified. Conversely, if you execute DROP FUNCTION followed by CREATE FUNCTION, execute privileges are reassigned.

- **Obfuscate a function definition** Use SET HIDDEN to obfuscate the definition of the associated function and cause it to become unreadable. The function can be unloaded and reloaded into other databases.

If SET HIDDEN is used, then debugging using the debugger does not show the function definition, nor is it available through procedure profiling.

Note *This setting is irreversible.* Retain the original function definition outside of the database.

- **Recompile a function** Use the RECOMPILE syntax to recompile a user-defined SQL function. When you recompile a function, the definition stored in the catalog is re-parsed and the syntax is verified. The preserved source for a function is not changed by recompiling. When you recompile a function, the definitions obfuscated by the SET HIDDEN clause remain obfuscated and unreadable.

Note For *required* parameters that accept variable names, an error is returned if one of the following conditions is true:

- The variable does not exist
- The contents of the variable are NULL
- The variable exceeds the length allowed by the parameter
- The data type of the variable does not match that required by the parameter

Privileges

You must be the owner of the function or have one of the following privileges:

- ALTER ANY PROCEDURE system privilege
- ALTER ANY OBJECT system privilege

To make a function external, you must have the CREATE EXTERNAL REFERENCE system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** ALTER FUNCTION is optional ANSI/ISO SQL Language Feature F381. However, in the SQL standard, ALTER FUNCTION cannot be used to re-define a SQL Persistent Stored Module (PSM) function definition. The ANSI/ISO SQL Standard does not include support for SET HIDDEN or RECOMPILE.

Example

1. In this example, MyFunction is created and altered. The SET HIDDEN clause obfuscates the function definition and makes it unreadable. To run this example, you must also have the CREATE PROCEDURE system privilege, since a function is being created before being altered.

```
CREATE FUNCTION MyFunction(  
    firstname CHAR(30),  
    lastname CHAR(30) )  
RETURNS CHAR(61)  
BEGIN  
    DECLARE name CHAR(61);  
    SET name = firstname || ' ' || lastname;  
    RETURN (name);  
ALTER FUNCTION MyFunction SET HIDDEN;  
END;
```

2. The following example creates and then alters a function using a variable in the NAMESPACE clause

- a. The following statements create a variable for a NAMESPACE clause:

```
CREATE VARIABLE @ns LONG VARCHAR ;  
SET @ns = 'http://wsdl.domain.com/' ;
```

- b. The following statement creates a function named FtoC that uses a variable in the

NAMESPACE clause:

```
CREATE FUNCTION FtoC ( IN temperature LONG VARCHAR )  
RETURNS LONG VARCHAR  
URL 'http://localhost:8082/FtoCService'  
TYPE 'SOAP:DOC'  
NAMESPACE @ns;
```

- c. The following statement alters the function FtoC so that it accepts and returns a FLOAT data type:

```
ALTER FUNCTION FtoC ( IN temperature FLOAT )  
RETURNS FLOAT  
URL 'http://localhost:8082/FtoCService'  
NAMESPACE @ns;
```

Related Information

[Hiding the contents of a procedure, function, trigger, event, or view](#)

Use the SET HIDDEN clause to obscure the contents of a procedure, function, trigger, event, or view.

[CREATE FUNCTION statement](#)

Creates a user-defined SQL function in the database.

[CREATE FUNCTION statement \[External call\]](#)

Creates an interface to a native or external function.

[CREATE FUNCTION statement \[Web service\]](#)

Creates a web client function that makes an HTTP or SOAP over HTTP request.

[ALTER PROCEDURE statement](#)

Modifies a procedure.

[DROP FUNCTION statement](#)

Removes a function from the database.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

ALTER INDEX statement

Renames an index, primary key, or foreign key, or changes the clustered nature of an index.

Syntax

```
ALTER { INDEX index-name  
| [ INDEX ] FOREIGN KEY role-name  
| [ INDEX ] PRIMARY KEY }  
ON [ owner.object-name { REBUILD | rename-clause | cluster-clause }
```

object-name : *table-name* | *materialized-view-name*

rename-clause : **RENAME { AS | TO }** *new-index-name*

cluster-clause : **CLUSTERED | NONCLUSTERED**

Parameters

rename-clause Specify the new name for the index, primary key, or foreign key.

When you rename the underlying index for a foreign or primary key, the corresponding RI constraint name for the index is not changed. However, the foreign key role name, if applicable, is the same as the index name and is changed. Use the ALTER TABLE statement to rename the RI constraint name, if necessary.

cluster-clause Specify whether the index should be changed to CLUSTERED or NONCLUSTERED. Only one index on a table can be clustered.

REBUILD clause Use this clause to rebuild an index, instead of dropping and recreating it.

Remarks

The ALTER INDEX statement carries out two tasks:

- It can be used to rename an index, primary key, or foreign key.
- It can be used to change an index type from nonclustered to clustered, or vice versa.

The ALTER INDEX statement can be used to change the clustering specification of the index, but does not reorganize the data. As well, only one index per table or materialized view can be clustered.

ALTER INDEX cannot be used to change an index on a local temporary table. An attempt to do so results in an Index not found error.

This statement cannot be executed when there are cursors opened with the WITH HOLD clause that use either statement or transaction snapshots.

Privileges

To alter an index on a table, you must be the owner of the table, or have one of the following privileges:

- REFERENCES privilege on the table
- ALTER ANY INDEX system privilege
- ALTER ANY OBJECT system privilege

To alter an index on a materialized view, you must be the owner of the materialized view, or have one of the following privileges:

- ALTER ANY INDEX system privilege
- ALTER ANY OBJECT system privilege

Side effects

Automatic commit. Closes all cursors for the current connection. If ALTER INDEX REBUILD is specified, a checkpoint is performed.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statement changes IX_product_name to be a clustered index:

```
ALTER INDEX IX_product_name ON GROUP0.Products  
CLUSTERED;
```

The following statement renames the index IX_product_name on the Products table to ixProductName:

```
ALTER INDEX IX_product_name ON GROUP0.Products  
RENAME TO ixProductName;
```

Related Information

[Snapshot isolation](#)

Snapshot isolation is designed to improve concurrency and consistency by maintaining different versions of data.

[CREATE INDEX statement](#)

Creates an index on a specified table or materialized view.

[ALTER TABLE statement](#)

Modifies a table definition or disables dependent views.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

SQL Anywhere 17 » SQL Anywhere Server - SQL Reference » SQL statements
 » Alphabetical list of SQL statements

ALTER LDAP SERVER statement

Alters an LDAP server configuration object.

Syntax

```
ALTER LDAP SERVER ldapua-server-name
[ ldapua-server-attrs ... ]
[ WITH { SUSPEND | ACTIVATE | REFRESH } ]
```


```
ldapua-server-attrs :
SEARCH DN search-dn-attributes ...
| AUTHENTICATION URL { 'url-string' | NULL }
| CONNECTION TIMEOUT timeout-value
| CONNECTION RETRIES retry-value
| TLS { ON | OFF }
```

```
search-dn-attributes :
URL { 'url-string' | NULL }
| ACCESS ACCOUNT { 'dn-string' | NULL }
| IDENTIFIED BY ( 'password' | NULL )
| IDENTIFIED BY ENCRYPTED { encrypted-password | NULL }
```

Parameters

SEARCH DN clause There is no default value for any parameter in the SEARCH DN clause.

- **URL** Use this clause to specify the host (by name or by IP address), port number, and search to be performed to do the lookup of the **LDAP Distinguished Name (DN)** for a given user ID. *url-string* is validated for correct LDAP URL syntax before it is stored in ISYSLDAPSERVER. The maximum size for this string is 1024 bytes.

The format of *url-string* must comply with the LDAP URL standard. See [The LDAP Standard Specification](#) .
- **ACCESS ACCOUNT** Use this clause to specify the LDAP Distinguished Name (DN) used by the database server to connect to the LDAP server. This is not a SQL Anywhere user, but a user created in the LDAP server specifically for logging in to the LDAP server. This user must have permissions within the LDAP server to search for DN's by user ID in the locations specified in the SEARCH DN URL clause. The maximum size for this string is 1024 bytes.
- **IDENTIFIED BY** Use this clause to specify the password associated with the user identified by ACCESS ACCOUNT. The maximum size is 255 bytes, and cannot be set to NULL.
- **IDENTIFIED BY ENCRYPTED** Use this clause to specify the password associated with the user identified by ACCESS ACCOUNT, provided in encrypted form, and is a binary value stored somewhere on disk. The maximum size of the binary is 289 bytes, and cannot be set to NULL. IDENTIFIED BY ENCRYPTED allows the password to be retrieved and used, without it becoming known.

AUTHENTICATION URL clause Use this clause to specify the host by name or IP address, and the port number of the LDAP server to use to authenticate a user. The DN of the user

obtained from a prior DN search and the user password are used to bind a new connection to the authentication URL. A successful connection to the LDAP server is considered proof of the identity of the connecting user. There is no default value for this parameter.

CONNECTION TIMEOUT clause Use this clause to specify the connection timeout, in milliseconds, to the LDAP server, both for searches for the DN and for authentication. The default value is 10 seconds.

CONNECTION RETRIES clause Use this clause to specify the number of retries for connections to the LDAP server, both for searches for the DN and for authentication. The valid range of values is 1-60. The default is 3.

TLS clause Use this clause to specify the use of the TLS protocol on connections to the LDAP server, both for the DN searches, and for authentication. The valid values are ON or OFF. The default is OFF. Use the Secure LDAP protocol by specifying `ldaps://` to begin the URL instead of `ldap://`. The TLS option must be set to OFF when using Secure LDAP.

WITH clause

- **WITH SUSPEND** Sets the state of the LDAP server communications to SUSPENDED (maintenance mode). The connections to the LDAP server are closed and authentication with the LDAP server is no longer performed.
- **WITH ACTIVATE** Activates the LDAP server for immediate use. This changes the state of the LDAP server communications to READY.
- **WITH REFRESH** Reinitializes LDAP user-authentication. This command does not change the state of the LDAP server if it is in the SUSPENDED state. When WITH REFRESH is specified for an LDAP server in the READY or ACTIVE state, connections to the LDAP server are closed. Then, the server option values are reread from the ISYSLDAPSERVER system table and are applied to new connections to the LDAP server and to incoming authentication requests to the database server.

Remarks

ALTER LDAP SERVER...WITH REFRESH is often used on an LDAP server that is in the ACTIVE or READY state to release any resources that may be held, or to reread changes made to files outside of the server, such as a change to the contents of the file specified by the `trusted_certificates_file` database option.

For other states, ALTER LDAP SERVER...WITH REFRESH has no effect.

If you use this statement in a procedure, do not specify the password (IDENTIFIED BY clause) as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

Privileges

You must have the MANAGE ANY LDAP SERVER system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example suspends the fictitious LDAP server named apps_primary.

```
ALTER LDAP SERVER apps_primary WITH SUSPEND;
```

The following example changes the LDAP server named apps_primary to use a different URL for authentication on host fairfax, port number 1066, sets connection retries to 10, and activates it.

```
ALTER LDAP SERVER apps_primary  
  AUTHENTICATION URL 'ldap://fairfax:1066/'  
  CONNECTION RETRIES 10  
  WITH ACTIVATE;
```

Related Information

[SYSLDAPSERVER system view](#)

The SYSLDAPSERVER system view contains one row for each LDAP server configuration object configured in the database. The underlying system table for this view is ISYSLDAPSERVER.

[CREATE LDAP SERVER statement](#)

Creates an LDAP server configuration object.

[DROP LDAP SERVER statement](#)

Drops an LDAP server configuration object.

[VALIDATE LDAP SERVER statement](#)

Validates an LDAP server configuration object.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

ALTER LOGIN POLICY statement

Alters an existing login policy.

Syntax

ALTER LOGIN POLICY *policy-name policy-options*

policy options :

policy-option [policy-option ...]

policy-option :

policy-option-name = policy-option-value

policy-option-value :

{ UNLIMITED

| DEFAULT

| legal-option-value }

Parameters

policy-name The name of the login policy. Specify root to modify the root login policy.

policy-option-name The name of the policy option.

policy-option-value The value assigned to the login policy option. If you specify UNLIMITED, no limits are used. If you specify DEFAULT, the default limits are used.

Remarks

When a login policy is altered, changes are immediately applied to all users.

If you do not specify a policy option, values for this login policy are taken from the root login policy. New policies do not inherit the MAX_NON_DBA_CONNECTIONS and ROOT_AUTO_UNLOCK_TIME policy options.

All new databases include a root login policy. You can modify the root login policy values, but you cannot delete the policy. An overview of the default values for the root login policy is provided in the table above.

Privileges

You must have the MANAGE ANY LOGIN POLICY system privilege.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example alters the fictitious Test1 login policy by changing the LOCKED and MAX_CONNECTIONS policy options. The LOCKED value indicates that users with the policy cannot establish new connections and the MAX_CONNECTIONS value limits the number of concurrent

connections that are allowed.

```
ALTER LOGIN POLICY Test1  
LOCKED=ON  
MAX_CONNECTIONS=5;
```

This example overrides the root login policy LOCKED and MAX_CONNECTIONS policy options.

```
ALTER LOGIN POLICY root  
LOCKED=ON  
MAX_CONNECTIONS=5;
```

The following example sets a primary and a secondary LDAP server for a fictitious ldap_user_policy login policy, and turns off the ability to failover to standard authentication, even when database option login_mode includes 'Standard'. This provides strict controls on users of this login policy so that only LDAP user authentication is used for authentication. In the event that a high volume of login connections occur such that the LDAP server is unable to respond and authenticate quickly, users whose retries and timeouts are exhausted will see connection failures to the database server rather than failover to use standard authentication.

```
ALTER LOGIN POLICY ldap_user_policy  
LDAP_PRIMARY_SERVER=ldapsrv1  
LDAP_SECONDARY_SERVER=ldapsrv2  
LDAP_FAILOVER_TO_STD=OFF;
```

The following example resets the timestamp value for a fictitious application_user_policy login policy to the current time. Any user that is assigned this policy have their Distinguished Name (DN) searched on the next login attempt, rather than using the value cached in ISYSUSER. This strategy purges old DN values held in ISYSUSER for users associated with this policy at the time of their next authentication.

```
ALTER LOGIN POLICY application_user_policy  
LDAP_REFRESH_DN=NOW;
```

Related Information

[Login policies](#)

A **login policy** consists of a set of rules that are applied when you create a database connection for a user.

[Altering a login policy](#)

Edit a login policy and change its options.

[ALTER USER statement](#)

Alters user settings.

[COMMENT statement](#)

Stores a comment for a database object in the system tables.

[CREATE LOGIN POLICY statement](#)

Creates a login policy.

[CREATE USER statement](#)

Creates a database user or group.

[DROP LOGIN POLICY statement](#)

Drops a login policy.

[DROP USER statement](#)

Drops a user.

[Login policy options and default values](#)

The following table lists the options that are governed by a login policy and includes the default values for the root login policy:

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

ALTER MATERIALIZED VIEW statement

Alters a materialized view.

Syntax

```
ALTER MATERIALIZED VIEW [ owner.]materialized-view-name {
  SET HIDDEN
| { ENABLE | DISABLE }
| { ENABLE | DISABLE } USE IN OPTIMIZATION
| { ADD PCTFREE percent-free-space | DROP PCTFREE }
| [ NOT ] ENCRYPTED
| [ { IMMEDIATE | MANUAL } REFRESH ]
}
```

percent-free-space : integer

Parameters

SET HIDDEN clause Use the SET HIDDEN clause to obfuscate the definition of a materialized view. *This setting is irreversible.*

ENABLE clause Use the ENABLE clause to enable a disabled materialized view, making it available for the database server to use. This clause has no effect on a view that is already enabled. After using this clause, you must refresh the view to initialize it, and recreate any text indexes that were dropped when the view was disabled.

DISABLE clause Use the DISABLE clause to disable use of the view by the database server. When you disable a materialized view, the database server drops the data and indexes for the view.

{ ENABLE | DISABLE } USE IN OPTIMIZATION clause Use this clause to specify whether you want the materialized view to be available for the optimizer to use. If you specify DISABLE USE IN OPTIMIZATION, the materialized view is used only when executing queries that explicitly reference the view. The default is ENABLE USE IN OPTIMIZATION.

ADD PCTFREE clause Specify the percentage of free space you want to reserve on each page. The free space is used if rows increase in size when the data is updated. If there is no free space on a page, every increase in the size of a row on that page requires the row to be split across multiple pages, causing row fragmentation and possible performance degradation.

The value of *percent-free-space* is an integer between 0 and 100. The value 0 specifies that no free space is to be left on each page. Each page is to be fully packed. A high value causes each row to be inserted into a page by itself. If PCTFREE is not set, or is dropped, the default PCTFREE setting is applied according to the database page size (200 bytes for a 4 KB page size, and 100 bytes for a 2 KB page size).

DROP PCTFREE clause Removes the PCTFREE setting currently in effect for the materialized view, and applies the default PCTFREE according to the database page size.

[NOT] ENCRYPTED clause Specify whether to encrypt the materialized view data. By default, materialized view data is not encrypted at creation time. To encrypt a materialized view, specify ENCRYPTED. To decrypt a materialized view, specify NOT ENCRYPTED.

REFRESH clause Use the REFRESH clause to change the refresh type for the materialized

view:

- **IMMEDIATE REFRESH** Use the IMMEDIATE REFRESH clause to change a manual view to an immediate view. The manual view must be valid and uninitialized to change the refresh type to IMMEDIATE REFRESH. If the view is in an initialized state, execute a TRUNCATE statement to change the state to uninitialized before executing the ALTER MATERIALIZED VIEW...IMMEDIATE REFRESH.
- **MANUAL REFRESH** Use the MANUAL REFRESH clause to change an immediate view to a manual view.

Remarks

If you alter a materialized view owned by another user, you must qualify the name by including the owner (for example, GROUPO.EmployeeConfidential). If you don't qualify the name, the database server looks for a materialized view with that name owned by you and alters it. If there isn't one, it returns an error.

When you disable a materialized view (DISABLE clause), it is no longer available for the database server to use for answering queries. As well, the data and indexes are dropped, and the refresh type changes to manual. Any dependent regular views are also disabled.

The DISABLE clause requires exclusive access not only to the view being disabled, but to any dependent views, since they are also disabled.

Table encryption must already be enabled on the database to encrypt a materialized view (ENCRYPTED clause). The materialized view is then encrypted using the encryption key and algorithm specified at database creation time.

The only operations a user can perform on a materialized view to change its data are refreshing, truncating, and disabling. However, immediate views are automatically updated by the database server. That is, once an immediate view is enabled and initialized, the database server maintains it automatically, without additional privileges checking.

Privileges

You must be the owner of the materialized view, or have the ALTER ANY MATERIALIZED VIEW or ALTER ANY OBJECT system privilege.

If you do not have a required privilege but want to alter a materialized view to be immediate (ALTER MATERIALIZED VIEW...IMMEDIATE REFRESH), you must own the view and all the tables it references.

Side effects

- Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statements creates the EmployeeConfid88 materialized view and then disables its use in optimization. To run this example you must also have the CREATE ANY MATERIALIZED VIEW system privilege, as well as SELECT privilege on the Employees and Departments tables.

Caution

When you are done with this example, you should drop the materialized view you

created. Otherwise, you cannot make schema changes to its underlying tables, Employees and Departments, when trying out other examples. You cannot alter the schema of a table that has enabled, dependent materialized view.

```
CREATE MATERIALIZED VIEW EmployeeConfid88 AS
  SELECT EmployeeID, Employees.DepartmentID, SocialSecurityNumber, Salary,
  ManagerID,
    Departments.DepartmentName, Departments.DepartmentHeadID
  FROM GROUP0.Employees, GROUP0.Departments
  WHERE Employees.DepartmentID=Departments.DepartmentID;
REFRESH MATERIALIZED VIEW EmployeeConfid88;
ALTER MATERIALIZED VIEW EmployeeConfid88 DISABLE USE IN OPTIMIZATION;
```

Related Information

[Materialized views](#)

A **materialized view** is a view whose result set has been precomputed from the base tables that it refers to and stored on disk, similar to a base table.

[View dependencies](#)

A view definition refers to other objects such as columns, tables, and other views, and these references make the view **dependent** on the objects to which it refers.

[Materialized views restrictions](#)

There are many restrictions when creating, initializing, refreshing, and using materialized views.

[Whether to set refresh type to manual or immediate](#)

There are two refresh types for materialized views: **manual** and **immediate**.

[Advanced: Status and properties for materialized views](#)

Materialized view availability and state can be determined from their status and properties.

[Enabling or disabling a materialized view](#)

Control whether a materialized view is available for querying by enabling and disabling it.

[Encrypting or decrypting a materialized view](#)

Encrypt materialized views for additional security on data.

[Hiding a materialized view definition](#)

Hide a materialized view definition from users. This obfuscates the view definition stored in the database.

[Enabling or disabling optimizer use of a materialized view](#)

Enable or disable optimizer use of a materialized view for satisfying queries.

[CREATE MATERIALIZED VIEW statement](#)

Creates a materialized view.

[REFRESH MATERIALIZED VIEW statement](#)

Initializes or refreshes the data in a materialized view by executing its query definition.

[DROP MATERIALIZED VIEW statement](#)

Removes a materialized view from the database.

[TRUNCATE statement](#)

Deletes all rows from a table without deleting the table definition.

[sa_refresh_materialized_views system procedure](#)

Initializes all materialized views that are in an uninitialized state.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

ALTER MIRROR SERVER statement

Modifies the attributes of a mirror server.

Syntax

- **Alter a mirror server**

```
ALTER MIRROR SERVER mirror-server-name
[AS { PRIMARY | MIRROR | ARBITER | PARTNER }]
[ server-option = { string | NULL } [ ... ] ]
```

- **Alter a mirror server as a copy**

```
ALTER MIRROR SERVER mirror-server-name
[AS COPY]
[ { FROM SERVER parent-name [ OR SERVER server-name ] | USING AUTO PARENT
} | ALTER PARENT FROM mirror-server-name ]
[ server-option = { string | NULL } [ ... ] ]
```

parent-name :
server-name | **PRIMARY**

server-option :
connection_string
logfile
preferred
state_file

Parameters

AS clause Use the AS clause to change the server-type of the mirror server from PARTNER to COPY or COPY to PARTNER. This clause is not needed and it is not recommended if you are not changing the type of mirror server.

- **PARTNER** Only a database server with the mirror server type of COPY can use this value to change its type to PARTNER. The parent definitions for the copy node are deleted.

The name of the mirror server must correspond to the name of the database server, as specified by the -n server option, and must match the value of the SERVER connection string parameter specified in the connection_string mirror server option.

In a database mirroring system, the partners use the connection string value to connect to each other. In a read-only scale-out system, the connection string is used by a copy-node that has this server as its parent.

- **COPY** Only a database server whose mirror server type is PARTNER can use this value to change its server type to a copy node. This partner server must also currently have the MIRROR role.

In a read-only scale-out system, this value specifies that the database server is a copy node. All connections to the database on this server are read-only. The name of the mirror server must correspond to the name of the database server, as specified by the -n server option, and must match the value of the SERVER connection string parameter specified in

the `connection_string` mirror server option.

FROM SERVER clause This clause can only be used when AS COPY is specified. This clause constructs a tree of servers for a scale-out system and indicates which servers the copy nodes obtain transaction log pages from.

The parent can be specified using the name of the mirror server or PRIMARY. An alternate parent for the copy node can be specified using the OR SERVER clause.

In a database mirroring system that has only two levels (partner and copy nodes), the copy nodes obtain transaction log pages from the current primary or mirror server.

A copy node determines which server to connect to by using its mirror server definition that is stored in the database. From its definition, it can locate the definition of its parent, and from its parent's definition, it can obtain the connection string to connect to the parent.

You do not have to explicitly define copy nodes for the scale-out system: you can choose to have the root node define the copy nodes when they connect.

USING AUTO PARENT clause This clause can only be used when AS COPY is specified. This clause causes the primary server to assign a parent for this server.

ALTER PARENT FROM clause This clause can only be used when AS COPY is specified. This clause changes the parent for this mirror server, and assigns all its siblings to be its children. The server name specified by the ALTER PARENT FROM clause is used to verify that the current parent for this server matches the value specified. This is used to ensure that only one of a collection of siblings is able to replace its parent if they all request the change simultaneously.

server-option clause You can specify a variable name for *server-option*.

The following options are also supported:

- **connection_string server option** Specifies the connection string to be used to connect to the server. The connection string for a mirror server should not include a user ID or password because they are not used when one mirror server connects to another mirror server.
- **logfile server option** Specifies the location of the file that contains one line per request that is sent between mirror servers if database mirroring is used. This file is used only for debugging.
- **preferred server option** Specifies whether the server is the preferred server in the mirroring system. You can specify either YES or NO. The preferred server assumes the role of primary server whenever possible. You specify this option when defining PARTNER servers.
- **state_file server option** Specifies the location of the file used for maintaining state information about the mirroring system. This option is required for database mirroring. In a mirroring system, a state file must be specified for servers with type PARTNER. For arbiter servers, the location is specified as part of the command to start the server.

Remarks

Read-only scale-out and database mirroring each require a separate license.

In a database mirroring system, the mirror server type can be PRIMARY, MIRROR, ARBITER, or PARTNER.

In a read-only scale-out system, the mirror server type can be PRIMARY, PARTNER, or COPY.

You can only change the mirror server type from COPY to PARTNER or from PARTNER to COPY. To

change to or from a PRIMARY, MIRROR, or ARBITER server type, you must drop the mirror server definition and recreate it.

Mirror server names for servers of type PARTNER and COPY must match the names of the database servers that will be part of the mirroring system (the name used with the -n server option). This requirement allows each database server to find its own definition and that of its parent. *mirror-server-name*, *parent-name*, and *server-name* above must contain only 7-bit ASCII characters.

When you convert from a copy node to a partner, the parent definitions are deleted from the mirror server definition.

To replace a mirror server definition, use the CREATE MIRROR SERVER statement with the OR REPLACE clause.

Note For parameters that accept variable names, an error is returned if one of the following conditions is true:

- The variable does not exist
- The contents of the variable are NULL
- The variable exceeds the length allowed by the parameter
- The data type of the variable does not match that required by the parameter

Privileges

You must have the MANAGE ANY MIRROR SERVER system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

1. The following example does the following:
 - a. Creates a primary server called scaleout_primary1.
 - b. Creates a copy node, scaleout_child1, for scaleout_primary1.
 - c. Creates a mirror server, scaleout_mirror1.
 - d. Using the ALTER MIRROR SERVER statement, reassigns scaleout_child1 to be a copy node of scaleout_mirror1.

```
CREATE MIRROR SERVER "scaleout_primary1"
  AS PRIMARY
  connection_string = 'server=scaleout_primary1;host=winxp-
2:6871,winxp-3:6872';

CREATE MIRROR SERVER "scaleout_child1"
  AS COPY FROM SERVER "scaleout_primary1"
  connection_string = 'server=scaleout_child1;host=winxp-2:6878';

CREATE MIRROR SERVER "scaleout_mirror1"
  AS MIRROR
  connection_string = 'server=scaleout_mirror1;host=winxp-2:6871,winxp-
3:6872';

ALTER MIRROR SERVER "scaleout_child1"
  FROM SERVER "scaleout_mirror1"
  connection_string = 'server=scaleout_child1;host=winxp-2:6878';
```

2. The following example alters a mirror server using a variable for the *connection_string* parameter.

- a. The following statement creates a variable for a connection string:

```
CREATE VARIABLE @connstr_value LONG VARCHAR
SET @connstr_value = ' server=new_scaleout_primary;host=winxp-2:6878 ';
```

- b. The following statement alters the mirror server new_scaleout_primary using the variable @connstr_value for the *connection_string* parameter:

```
ALTER MIRROR SERVER new_scaleout_primary AS PRIMARY
connection_string = @connstr_value = @connstr_value;
```

Related Information

[How child copy nodes are added](#)

The child_creation option of the SET MIRROR OPTION statement controls how child nodes are added to a read-only scale-out system.

[Connection parameters](#)

Connection parameters are included in connection strings.

[Troubleshooting: State information files of the partners and arbiter](#)

The partners and the arbiter in the high availability system each maintain a state information file that records that server view of the state of the mirroring system.

[Automatically assign the parent of a copy node](#)

The SET MIRROR OPTION statement supports two options that can be used to assign new copy nodes to a parent in the tree so that the work of distributing transaction log pages is balanced among the nodes.

[Preferred database server in a database mirroring system](#)

In a database mirroring system, you can specify one partner server as the preferred server.

[Separately licensed components](#)

These components are licensed separately and may need to be ordered separately if not included in your edition of SQL Anywhere.

[Converting a partner server to a copy node](#)

Convert a partner server in a database mirroring system to a copy node in a read-only scale-out system without stopping the system.

[SYSMIRRORSERVER system view](#)

The underlying system table for this view is ISYSMIRRORSERVER.

[CREATE MIRROR SERVER statement](#)

Creates or replaces a mirror server that is being used for database mirroring or read-only scale-out.

[COMMENT statement](#)

Stores a comment for a database object in the system tables.

[DROP MIRROR SERVER statement](#)

Drops a mirror server.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

ALTER ODATA PRODUCER statement

Alters an OData Producer.

Syntax

ALTER ODATA PRODUCER *name* [*producer-clause...*]

producer-clause:

[**ADMIN USER** { **NULL** | *user* }]

[**AUTHENTICATION** [**DATABASE** | **USER** *user*]]

[[**NOT**] **ENABLED**]

[**MODEL** [**FROM**] { **FILE** *string-or-variable* [**ENCODING** *string*] | **VALUE** *string-or-variable*

[**ENCODING** *string*] }]

[[**SERVICE**] **ROOT** *string*]

[**USING** { *string* | **NULL** }]

Parameters

- **ADMIN USER clause** Specifies an administrative user that the OData Producer uses to create tables, a procedure, and an event to manage repeatable requests. The user must have the CREATE TABLE, CREATE PROCEDURE, MANAGE ANY EVENT, and VERIFY ODATA system privileges. This user cannot be the same user specified by the AUTHENTICATION clause.
- **AUTHENTICATION clause** Specifies how the OData server connects to the database. Specify DATABASE for users to connect with personalized credentials. These credentials are requested using Basic HTTP authentication. Specify USER for all users to connect using the connection string of the specified user. The specified user must have the VERIFY ODATA system privilege.
- **ENABLED clause** Specifies whether this OData Producer is enabled or disabled. By default, OData Producers are enabled.
- **MODEL clause** Specifies the OData Producer service model (given in OSDL) that indicates which tables and views are exposed in the OData metadata. If you specify the FILE clause, then the value must be a variable or string of the file name that contains the OSDL data. If the file name is not fully qualified, then the path is relative to the database server's current working directory. If the file name is specified, then the file is read and the contents of the file are stored in the database and logged in the transaction log. By default, tables and views are exposed based on user privileges. Tables and views without primary keys are not exposed.

Use the ENCODING clause to specify the character set of the model data or file. The default is UTF-8.
- **SERVICE ROOT clause** Specifies the root of the OData service on the OData server. All resources for the OData Producer are accessed by using URIs of the following format:
`scheme:host:port/path-prefix/resource-path[query-options]`.
The SERVICE ROOT clause is /path-prefix. The value cannot be NULL and must start with /.
- **USING clause** This string specifies additional OData Producer options. Options are specified in a semicolon-separated list of name=value pairs.

Option	Description
--------	-------------

Option	Description
ConnectionPoolMaximum = <i>num-max-connections</i>	<p>Indicates the maximum number of simultaneous connections that this OData Producer keeps open for use in the connection pool.</p> <p>Fewer connections might be used by the connection pool depending on the OData server load.</p> <p>By default, the connection pool size is limited to half of the maximum number of simultaneous connections that the database server supports.</p>
CSRFTokenTimeout = <i>num-seconds-valid</i>	<p>Enables CSRF token checking and specifies the number of seconds that a token is valid for.</p> <p>By default, this value is 0, which disables CSRF token checking. Otherwise, the number of seconds must be a valid integer value from 1 to 1800.</p>
PageSize = <i>num-max-entities</i>	<p>Specifies the maximum number of entities to include in a retrieve entity set response before issuing a next link.</p> <p>The default setting is 100.</p>
ReadOnly = { true false }	<p>Indicates whether modification requests should be ignored.</p> <p>The default setting is false.</p>
RepeatRequestForDays = { <i>days-number</i> }	<p>Specifies how long repeatable requests are valid.</p> <p>This value must be an integer ranging from 1 to 31.</p> <p>The default setting is 2.</p> <p>This option is only effective when the ADMIN USER clause is not NULL.</p>
SecureOnly = { true false }	<p>Indicates whether the Producer should only listen for requests on the HTTPS port.</p> <p>The default setting is false.</p>
ServiceOperationColumnNames = { generate database }	<p>Specifies whether the column names in the metadata should be generated or taken from the result set columns in the database when naming the properties of the ComplexType used in the ReturnType.</p> <p>The default setting is generate.</p>

Remarks

Alters an OData Producer that runs as a sub-process of the database server.

Privileges

You must have the MANAGE ODATA system privilege.

If you specify the AUTHENTICATION USER clause or the ADMIN USER clause, then the specified users must have the VERIFY ODATA system privilege.

Side effects

If the OData Producer is running, then it may be stopped and restarted.

Example

Assume that you want to create an OData Producer that accesses objects owned by user Dave through a URL beginning with `/odata/dave` and that you have a model file called `dave.txt` that contains details of the tables that you want to expose. You also want to ensure that no data can be updated through the OData interface and that requests are repeated for up to ten days. Create an OData Producer called `dave_producer` by executing the following statement:

```
CREATE ODATA PRODUCER dave_producer
MODEL FROM FILE 'dave.txt'
SERVICE ROOT '/odata/dave'
AUTHENTICATION USER dave
USING 'ReadOnly=true;RepeatRequestForDays=10';
```

If you then want to temporarily disable this OData Producer, execute the following statement:

```
ALTER ODATA PRODUCER dave_producer NOT ENABLED;
```

To remove the read-only restriction but retain the repeatable request limit, and re-enable the OData Producer, execute the following statement:

```
ALTER ODATA PRODUCER dave_producer
ENABLED
USING 'ReadOnly=false;RepeatRequestForDays=10';
```

Related Information

[OData support](#)

OData (Open Data Protocol) enables data services over RESTful HTTP. It allows you to perform operations through URIs (Uniform Resource Identifiers) to access and modify information.

[Network protocol options](#)

Network protocol options enable you to work around traits of different network protocol implementations.

[CREATE ODATA PRODUCER statement](#)

Creates or replaces an OData Producer.

[DROP ODATA PRODUCER statement](#)

Drops an OData Producer.

[COMMENT statement](#)

Stores a comment for a database object in the system tables.

[-xs database server option](#)

Specifies server-side web services communications protocols.

[SYSODATAPRODUCER system view](#)

Each row of the SYSODATAPRODUCER system view describes an OData Producer. The underlying system table for this view is ISYSODATAPRODUCER.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

ALTER PROCEDURE statement

Modifies a procedure.

Syntax

- **Change the definition of a procedure**

ALTER PROCEDURE [*owner.*]*procedure-name* *procedure-definition*

procedure-definition : [see the CREATE PROCEDURE statement](#)

- **Obfuscate a procedure definition**

ALTER PROCEDURE [*owner.*]*procedure-name*
SET HIDDEN

- **Recompile a procedure**

ALTER PROCEDURE [*owner.*]*procedure-name*
RECOMPILE

Remarks

The ALTER PROCEDURE statement must include the entire new procedure. You can use PROC as a synonym for PROCEDURE.

- **Change the definition of a procedure** The ALTER PROCEDURE statement is identical in syntax to the CREATE PROCEDURE statement except for the first word. Both Watcom and Transact-SQL dialect procedures can be altered through the use of ALTER PROCEDURE.

With ALTER PROCEDURE, existing privileges on the procedure are not changed. If you execute DROP PROCEDURE followed by CREATE PROCEDURE, execute privileges are reassigned.

- **Obfuscate a procedure definition** Use SET HIDDEN to obfuscate the definition of the associated procedure and cause it to become unreadable. The procedure can be unloaded and reloaded into other databases.

If SET HIDDEN is used, debugging using the debugger does not show the procedure definition, and the definition is not available through procedure profiling.

Note *This change is irreversible.* It is recommended that you retain the original procedure definition outside of the database.

- **Recompile a procedure** Use the RECOMPILE syntax to recompile a stored procedure. When you recompile a procedure, the definition stored in the catalog is re-parsed and the syntax is verified. For procedures that generate a result set but do not include a RESULT clause, the database server attempts to determine the result set characteristics for the procedure and stores the information in the catalog. This can be useful if a table referenced by the procedure has been altered to add, remove, or rename columns since the procedure was created.

The procedure definition is not changed by recompiling. You can recompile procedures with definitions hidden with the SET HIDDEN clause, but their definitions remain hidden.

Note For *required* parameters that accept variable names, an error is returned if one of the

following conditions is true:

- The variable does not exist
- The contents of the variable are NULL
- The variable exceeds the length allowed by the parameter
- The data type of the variable does not match that required by the parameter

Privileges

You must be the owner of the procedure or have one of the following privileges:

- ALTER ANY PROCEDURE system privilege
- ALTER ANY OBJECT system privilege

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** ALTER PROCEDURE is optional ANSI/ISO SQL Language Feature F381. However, in the SQL standard, ALTER PROCEDURE cannot be used to re-define a stored procedure definition, and Transact-SQL dialect procedures are not supported. The ANSI/ISO SQL Standard does not include support for SET HIDDEN or RECOMPILE.

Example

1. The following example creates and then alters a procedure using a variable in the NAMESPACE clause
 - a. The following statements create a variable for a NAMESPACE clause:

```
CREATE VARIABLE @ns LONG VARCHAR  
SET @ns = 'http://wsdl.domain.com/';
```

- b. The following statement creates a procedure named FtoC that uses a variable in the NAMESPACE clause:

```
CREATE PROCEDURE FtoC ( IN temperature LONG VARCHAR )  
URL 'http://localhost:8082/FtoCService'  
TYPE 'SOAP:DOC'  
NAMESPACE @ns;
```

- c. The following statement alters the procedure FtoC so that the temperature parameter accepts a FLOAT data type:

```
ALTER PROCEDURE FtoC ( IN temperature FLOAT )  
URL 'http://localhost:8082/FtoCService'  
NAMESPACE @ns;
```

Related Information

[Hiding the contents of a procedure, function, trigger, event, or view](#)

Use the SET HIDDEN clause to obscure the contents of a procedure, function, trigger, event, or view.

[CREATE PROCEDURE statement](#)

Creates a user-defined SQL procedure in the database.

[CREATE PROCEDURE statement \[External call\]](#)

Creates an interface to a native or external procedure.

[CREATE PROCEDURE statement \[Web service\]](#)

Creates a user-defined web client procedure that makes HTTP or SOAP requests to an HTTP server.

[ALTER FUNCTION statement](#)

Modifies a function.

[DROP PROCEDURE statement](#)

Removes a procedure from the database.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

ALTER PUBLICATION statement [MobiLink] [SQL Remote]

Alters a publication. In MobiLink, a publication identifies synchronized data in a SQL Anywhere remote database. In SQL Remote, a publication identifies replicated data in both consolidated and remote databases.

Syntax

ALTER PUBLICATION [*owner.*] *publication-name* *alterpub-clause*, ...

alterpub-clause :

ADD *article-definition*

| **ALTER** *article-definition*

| { **DELETE** | **DROP** } **TABLE** [*owner.*] *table-name*

| **RENAME** *publication-name*

article-definition :

TABLE *table-name* [(*column-name*, ...)]

[**WHERE** *search-condition*]

[**SUBSCRIBE BY** *expression*]

[**USING** ([**PROCEDURE**] [*owner.*] *procedure-name*]

FOR UPLOAD { **INSERT** | **DELETE** | **UPDATE** }, ...)]

Remarks

This statement is applicable only to MobiLink and SQL Remote.

The contribution to a publication from one table is called an article. 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 definition of the modified article must be entered.

It is recommended that you perform a successful synchronization of a publication immediately before you alter it.

You cannot use the WHERE clause for publications that are defined as FOR DOWNLOAD ONLY or WITH SCRIPTED UPLOAD.

The SUBSCRIBE BY clause applies to SQL Remote only.

The USING clause is for scripted upload only.

You set options for a MobiLink publication with the ADD OPTION clause in the ALTER SYNCHRONIZATION SUBSCRIPTION statement or CREATE SYNCHRONIZATION SUBSCRIPTION statement.

When altering a MobiLink publication, an article can only be dropped after the execution of a START SYNCHRONIZATION SCHEMA CHANGE statement.

Requires exclusive access to all tables referred to in the statement as well as all tables in publication being modified.

Privileges

You must be the owner of the publication, or have one of the following privileges:

- ALTER privilege on the publication

- **SYS_REPLICATION_ADMIN_ROLE** system role

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statement adds the Customers table to the pub_contact publication.

```
ALTER PUBLICATION pub_contact  
ADD TABLE GROUP0.Customers;
```

Related Information

[Publications](#)

A publication is a database object that identifies the data that is to be synchronized. It defines the data to be uploaded, and it limits the tables that can be downloaded to. (The download is defined in the download_cursor script.)

[Publications and articles](#)

A **publication** defines the set of data to be replicated. A publication can include data from several database tables.

[CREATE PUBLICATION statement \[MobiLink\] \[SQL Remote\]](#)

Creates a publication. In MobiLink, a publication identifies synchronized data in a SQL Anywhere remote database. In SQL Remote, publications identify replicated data in both consolidated and remote databases.

[DROP PUBLICATION statement \[MobiLink\] \[SQL Remote\]](#)

Drops a publication.

[ALTER SYNCHRONIZATION SUBSCRIPTION statement \[MobiLink\]](#)

Alters the properties of a synchronization subscription in a SQL Anywhere remote database.

[CREATE SYNCHRONIZATION SUBSCRIPTION statement \[MobiLink\]](#)

Creates a subscription in a SQL Anywhere remote database between a MobiLink user and a publication.

[SYSSYNC system view](#)

The SYSSYNC system view contains information relating to synchronization.

[START SYNCHRONIZATION SCHEMA CHANGE statement \[MobiLink\]](#)

Starts a MobiLink synchronization schema change.

[Changes to avoid on a running system](#)

There are a few limitations to be aware of if you are making changes to deployed and running SQL Remote systems.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

ALTER REMOTE MESSAGE TYPE statement [SQL Remote]

Changes the publisher's message system, or the publisher's address for a given message system, for a message type that has been created.

Syntax

ALTER REMOTE MESSAGE TYPE *message-system*
ADDRESS *address*

message-system : **FILE** | **FTP** | **SMTP**

address : *string*

Parameters

message-system One of the message systems supported by SQL Remote. It must be one of the following values: FILE, FTP, or SMTP.

address A string containing a valid address for the specified message system.

Remarks

The statement changes the publisher's address for a given message type.

The Message Agent sends outgoing messages from a database by one of the supported message links. The Extraction utility uses this address when it executes the GRANT CONSOLIDATE statement in the remote database.

The address is the publisher's address under the specified message system. If it is an email system, the address string must be a valid email 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.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statement changes the publisher's address for the FILE message link to new_addr.

```
ALTER REMOTE MESSAGE TYPE file  
ADDRESS 'new_addr';
```

Related Information

[SQL Remote message systems](#)

In SQL Remote replication, a message system is a protocol for exchanging messages between the consolidated database and a remote database. SQL Remote exchanges data among databases using one or more underlying message systems.

[Altering a message type \(SQL Central\)](#)

Alter message types to change the address of a publisher. You cannot change the name of an existing message type; instead, you must delete it and create a new message type with the new name.

[CREATE REMOTE \[MESSAGE\] TYPE statement \[SQL Remote\]](#)

Identifies a message link and return address for outgoing messages from a database.

[GRANT CONSOLIDATE statement \[SQL Remote\]](#)

Identifies the database immediately above the current database in a SQL Remote hierarchy, that will receive messages from the current database.

[SQLREMOTE environment variable](#)

Specifies subdirectories that are addresses for the SQL Remote FILE message link.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

ALTER ROLE statement

Migrates a compatibility role to a user-defined role, and then drops the compatibility role.

Syntax

ALTER ROLE *compatibility-role-name*

MIGRATE TO *new-role-name* [, *new-sa-role-name*, *new-sso-role-name*]

Parameters

- **compatibility-role-name** Use this parameter to specify the name of the compatibility role you are migrating.
- **new-role-name** Use this parameter to specify the name of the new role you are creating.
- **new-sa-role-name** Use this parameter to specify the name of the new role to migrate the SYS_AUTH_SA_ROLE role to. This parameter is required when migrating SYS_AUTH_DBA_ROLE, which causes SYS_AUTH_SA_ROLE to be migrated automatically.
- **new-sso-role-name** Use this parameter to specify the name of the new role to migrate the SYS_AUTH_SSO_ROLE role to. This parameter is required when migrating SYS_AUTH_DBA_ROLE, which causes SYS_AUTH_SSO_ROLE to be migrated automatically.

Remarks

The name of the new role must not begin and end with 'SYS_' and '_ROLE', respectively. For example SYS_MyBackup_ROLE is not an acceptable name for a user-defined role, whereas MyBackup_ROLE and SYS_MyBackup are acceptable.

When you execute the ALTER ROLE statement, grantees of the compatibility role are granted the new role.

You can restore migrated compatibility roles that have been migrated and then dropped by executing a CREATE ROLE statement and specifying the compatibility role name. For example, [CREATE ROLE SYS_AUTH_BACKUP_ROLE](#); restores the SYS_AUTH_BACKUP_ROLE compatibility role.

Initially, only users with full administration rights (DBAs) can administer the new role, but you can use the CREATE ROLE statement with the OR REPLACE clause to specify additional administrators.

Use the GRANT or REVOKE statements to grant system privileges to the role, or revoke system privileges from the role.

You can migrate the SYS_AUTH_SA_ROLE and SYS_AUTH_SSO_ROLE compatibility roles by migrating SYS_AUTH_DBA_ROLE compatibility, which causes SYS_AUTH_SA_ROLE and SYS_AUTH_SSO_ROLE to be migrated automatically. When migrating SYS_AUTH_DBA_ROLE, you must include the *new-sa-role-name* and *new-sso-role-name* parameters to give new names to migrated SYS_AUTH_SA_ROLE and SYS_AUTH_SSO_ROLE roles.

Privileges

You must have the MANAGE ROLES system privilege and administrative rights on the compatibility role you are migrating.

Side effects

None

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statement migrates all users and underlying system privileges granted to SYS_AUTH_BACKUP_ROLE role to a new role, custom_Backup_ROLE, and then drops SYS_AUTH_BACKUP_ROLE from the database.

```
ALTER ROLE SYS_AUTH_BACKUP_ROLE  
MIGRATE TO custom_Backup_ROLE;
```

The following statement migrates all users, underlying system privileges, and roles granted to SYS_AUTH_DBA_ROLE compatibility role to a new role, custom_DBA. It then automatically migrates all users, underlying system privileges, and roles granted to SYS_AUTH_SA_ROLE and SYS_AUTH_SSO_ROLE to new roles called custom_SA and custom_SSO, respectively. Finally, it drops SYS_AUTH_DBA_ROLE, SYS_AUTH_SA_ROLE, and SYS_AUTH_SSO_ROLE from the database.

```
ALTER ROLE SYS_AUTH_DBA_ROLE  
MIGRATE TO custom_DBA, custom_SA, custom_SSO;
```

Related Information

Roles

There are three types of roles in the role-based security model: **system roles**, **user-defined roles** (which include user-extended roles), and **compatibility roles**.

[CREATE ROLE statement](#)

Creates or replaces a role, creates a user-extended role, or modifies administrators for a role.

[min_role_admins option](#)

Sets the minimum number of administrators required to administer roles.

[REVOKE ROLE statement](#)

Revokes roles and privileges from users and roles.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

ALTER SEQUENCE statement

Alters a sequence.

Syntax

```
ALTER SEQUENCE [owner.]sequence-name  
[ RESTART WITH signed-integer ]  
[ INCREMENT BY signed-integer ]  
[ MINVALUE signed-integer | NO MINVALUE ]  
[ MAXVALUE signed-integer | NO MAXVALUE ]  
[ CACHE integer | NO CACHE ]  
[ CYCLE | NO CYCLE ]
```

Parameters

RESTART WITH clause Restarts the named sequence with the specified value.

INCREMENT BY clause Defines the amount the next sequence value is incremented from the last value assigned. The default is 1. Specify a negative value to generate a descending sequence. An error is returned if the INCREMENT BY value is 0.

MINVALUE clause Defines the smallest value generated by the sequence. The default is 1. An error is returned if MINVALUE is greater than $(2^{63}-1)$ or less than $-(2^{63}-1)$. An error is also returned if MINVALUE is greater than MAXVALUE.

MAXVALUE clause Defines the largest value generated by the sequence. The default is $2^{63}-1$. An error is returned if MAXVALUE is greater than $2^{63}-1$ or less than $-(2^{63}-1)$.

CACHE clause Specifies the number of preallocated sequence values that are kept in memory for faster access. When the cache is exhausted, the sequence cache is repopulated and a corresponding entry is written to the transaction log. At checkpoint time, the current value of the cache is forwarded to the ISYSEQUENCE system table. The default is 100.

CYCLE clause Specifies whether values should continue to be generated after the maximum or minimum value is reached.

Remarks

If the named sequence cannot be located, an error message is returned.

Privileges

You must be the owner of the sequence, or have one of the following privileges:

- ALTER ANY SEQUENCE system privilege
- ALTER ANY OBJECT system privilege

Side effects

None

Standards

- **ANSI/ISO SQL Standard** The ALTER SEQUENCE statement is part of optional ANSI/ISO SQL Language Feature T176. The CACHE clause is not in the standard.

Example

The following example sets a new maximum value for a sequence named Test:

```
ALTER SEQUENCE Test  
  MAXVALUE 1500;
```

Related Information

[Use of a sequence to generate unique values](#)

You can use a **sequence** to generate values that are unique across multiple tables or that are different from a set of natural numbers.

[CREATE SEQUENCE statement](#)

Creates a sequence that can be used to generate primary key values that are unique across multiple tables, and for generating default values for a table.

[DROP SEQUENCE statement](#)

Drops a sequence.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

ALTER SERVER statement

Modifies the attributes of a remote server.

Syntax

- **Alter a remote server**

```
ALTER [ REMOTE ] SERVER server-name
[ CLASS server-class | variable ]
[ USING connection-string-info | variable ]
[ CAPABILITY cap-name { ON | OFF | VALUE variable } ]
[ READ ONLY [ ON | OFF | VALUE variable ] ]
[ DEFAULT LOGIN string | variable [ IDENTIFIED BY string | variable ] | NO DEFAULT LOGIN ]
```

server-class-string :

```
{ 'ADSODBC' | 'ADS_ODBC' }
| 'ASEODBC' | 'ASE_ODBC' }
| 'DB2ODBC' | 'DB2_ODBC' }
| 'HANAODBC' | 'HANA_ODBC' }
| 'IQODBC' | 'IQ_ODBC' }
| 'MIRROR'
| 'MSACCESSODBC' | 'MSACCESS_ODBC' }
| 'MSSODBC' | 'MSS_ODBC' }
| 'MYSQLODBC' | 'MYSQL_ODBC' }
| 'ODBC'
| 'ORAODBC' | 'ORA_ODBC' }
| 'SAODBC' | 'SA_ODBC' }
| 'ULODBC' | 'UL_ODBC' }
```

connection-info-string :

```
{ 'data-source-name' | 'sqlanywhere-connection-string' }
```

- **Alter a directory access server**

```
ALTER SERVER server-name
[ CLASS 'DIRECTORY' ]
[ USING using-string | variable ]
[ CAPABILITY cap-name { ON | OFF | VALUE variable } ]
[ READ ONLY [ ON | OFF | VALUE variable ] ]
[ ALLOW { 'ALL' | 'SPECIFIC' | variable } USERS ]
```

using-string :

```
'ROOT = path [ ;SUBDIRS = n ] [ ;CREATEDIRS = { YES | NO } ] [ ;DELIMITER =
{ / | \ } ]'
```

- **Alter a remote server (SAP HANA syntax)**

```

ALTER REMOTE SOURCE remote-source-name
  ADAPTER adapter-name | variable
  CONFIGURATION connection-info-string | variable
  [ CAPABILITY cap-name { ON | OFF | VALUE variable } ]
  [ READ ONLY [ ON | OFF | VALUE variable ] ]
  [ WITH CREDENTIAL TYPE { 'PASSWORD' | variable } USING { 'USER=remote-user;password=remote-password' | variable } | WITH NO CREDENTIAL ]

```

Parameters

ALTER [REMOTE] SERVER The REMOTE keyword is optional when altering a remote server and is provided for compatibility with other databases.

CLASS clause Specify this clause to change the server class.

USING clause Specify the server connection information.

The string in the USING clause can contain local or global variable names enclosed in braces ({ *variable-name* }). The SQL variable name must be of type CHAR, VARCHAR, or LONG VARCHAR. For example, a USING clause that contains '*DSN*={@*mydsn*}' indicates that @*mydsn* is a SQL variable and that the current value of the @*mydsn* variable is substituted when a connection is made to the remote data access server.

CAPABILITY clause Specify this clause to turn a server capability ON or OFF. Server capabilities are stored in the ISYSCAPABILITY system table. The names of these capabilities are accessible via the SYSCAPABILITYNAME system view. The ISYSCAPABILITY system table and SYSCAPABILITYNAME system view are not populated with data until the first connection to a remote server is made. For subsequent connections, the database server's capabilities are obtained from the ISYSCAPABILITY system table.

In general, you do not need to alter a server's capabilities. It may be necessary to alter the capabilities of a generic server of class ODBC.

READ ONLY clause (remote server) Specifies whether the remote server is accessed in read-only mode.

If the READ ONLY clause is not specified, then the read-only setting of the server remains unaffected. If READ ONLY or READ ONLY ON is specified, then the server is changed to be read only. If READ ONLY OFF is specified, then the server is changed to be read-write.

READ ONLY clause (directory access server) Specifies whether the files accessed by the directory are read-only and cannot be modified. By default, READ ONLY is set to NO.

ALLOW USERS clause Specifies whether users can use the directory access server without requiring an external login (externlogin). Specifying the ALLOW 'ALL' USERS clause allows you to create or alter directory access servers to no longer require external logins for each user. Specifying the ALLOW 'SPECIFIC' USERS is equivalent to requiring an external login for each user that uses the directory access server. Not specifying this clause is equivalent to specifying ALLOW 'SPECIFIC' USERS.

DEFAULT LOGIN clause Specify DEFAULT LOGIN to add or change the default login for the remote server. Specifying NO DEFAULT LOGIN removes the default login for the remote server, if one exists.

ALTER REMOTE SOURCE (SAP HANA syntax) This syntax, including the parameters and their values, is semantically equivalent to the syntax for altering a remote server (CREATE [REMOTE]) syntax, and is provided for compatibility with SAP HANA servers. There is a one-to-one

clause match between the two syntaxes as follows:

- **ADAPTER *adapter-name*** See the CLASS clause description for the ALTER [REMOTE] SERVER syntax.
- **CONFIGURATION *connection-info-string*** See the USING clause description for the ALTER [REMOTE] SERVER syntax.
- **CAPABILITY *cap-name*** See the CAPABILITY clause description for the ALTER [REMOTE] SERVER syntax.
- **READ ONLY [ON | OFF] clause** See the READ ONLY clause description for the ALTER [REMOTE] SERVER syntax (remote server).
- **WITH CREDENTIAL TYPE clause** See the DEFAULT LOGIN clause description for the ALTER [REMOTE] SERVER syntax. If you are using variables in the WITH CREDENTIAL TYPE clause, then the variable must be a string containing the value 'PASSWORD' and the USING variable must be string using the 'user=...;password=...' format.

Remarks

Changes do not take effect until the next connection to the remote server.

If you use this statement in a procedure, do not specify the password (IDENTIFIED BY clause) as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

Note For *required* parameters that accept variable names, the database server returns an error if any of the following conditions is true:

- The variable does not exist
- The contents of the variable are NULL
- The variable exceeds the length allowed by the parameter
- The data type of the variable does not match that required by the parameter

Privileges

You must have the SERVER OPERATOR system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example changes the server class of the Adaptive Server Enterprise server named ase_prod so its connection to SQL Anywhere is ODBC based. Its data source name is ase_datasource.

```
ALTER SERVER ase_prod
CLASS 'ASEODBC'
USING 'ase_datasource';
```

The following example changes the server class of the Adaptive Server Enterprise server named `ase_prod` so its connection to SQL Anywhere is ODBC based. Its data source name is obtained from the variable `ase_source`.

```
ALTER SERVER ase_prod
CLASS 'ASEODBC'
USING '{ase_source}';

CREATE VARIABLE ase_source VARCHAR(128);
SET ase_source = 'ase_datasource';
```

The following example changes a capability of a server `ase_prod`.

```
ALTER SERVER ase_prod
CAPABILITY 'insert select' OFF;
```

The following example alters a directory access server so that it retrieves nine levels of subdirectories within the directory `c:\temp`.

```
ALTER SERVER ase_prod
CLASS 'DIRECTORY'
USING 'ROOT=c:\\temp;SUBDIRS=9';
```

Related Information

[Remote data access](#)

Remote data access gives you access to data in other data sources as well as access to the files on the computer that is running the database server.

[Remote servers and remote table mappings](#)

SQL Anywhere presents tables to a client application as if all the data in the tables were stored in the database to which the application is connected. Before you can map remote objects to a local proxy table, you must define the remote server where the remote object is located.

[Directory access servers](#)

A **directory access server** is a remote server that gives you access to the local file structure of the computer running the database server.

[CREATE SERVER statement](#)

Creates a remote server or a directory access server.

[DROP REMOTE CONNECTION statement](#)

Drops remote data access connections to a remote server.

[DROP SERVER statement](#)

Drops a remote server from the catalog.

[SYSCAPABILITY system view](#)

Each row of the SYSCAPABILITY system view specifies the status of a capability on a remote database server. The underlying system table for this view is ISYSCAPABILITY.

[SYSCAPABILITYNAME system view](#)

Each row in the SYSCAPABILITYNAME system view provides a name for each capability ID in the SYSCAPABILITY system view.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

ALTER SERVICE statement [HTTP web service]

Alters an existing HTTP web service.

Syntax

```
ALTER SERVICE service-name
[ TYPE { 'RAW' | 'HTML' | 'JSON' | 'XML' } ]
[ URL [ PATH ] { ON | OFF | ELEMENTS } ]
[ common-attributes ]
[ AS { statement | NULL } ]
```

common-attributes :

```
[ AUTHORIZATION { ON | OFF } ]
[ ENABLE | DISABLE ]
[ METHODS 'method,...' ]
[ SECURE { ON | OFF } ]
[ USER { user-name | NULL } ]
```

method :

```
DEFAULT
| POST
| GET
| HEAD
| PUT
| DELETE
| NONE
| *
```

Parameters

service-name Web service names can be any sequence of alphanumeric characters or slash (/), hyphen (-), underscore (_), period (.), exclamation mark (!), tilde (~), asterisk (*), apostrophe ('), left parenthesis ((), or right parenthesis ()), except that the service name must not begin or end with a slash (/) or contain two or more consecutive slashes (for example, //).

You can name your service root, but this name has a special function.

TYPE clause Identifies the type of the service where each service defines a specific response format. The type must be one of the listed service types. There is no default value.

- o **'RAW'** The result set is sent to the client without any formatting. Utilization of this service requires that all content markup is explicitly provided. Complex dynamic content containing current content with markup, JavaScript and images can be generated on demand. The media type may be specified by setting the Content-Type response header using the `sa_set_http_header` procedure. Setting the Content-Type header to 'text/html' is good practice when generating HTML markup to ensure that all browsers display the markup as HTML and not text/plain.
- o **'HTML'** The result set is returned as an HTML representation of a table or view.
- o **'JSON'** The result set is returned in JavaScript Object Notation (JSON). A JSON service does not automatically process JSON input. It only presents data (in the response) in JSON

format. JSON accepts POST/PUT methods where **application/x-www-form-urlencoded** is supported. If for a POST/PUT method, **Content-Type: application/json** is specified, then the application may use `http_variable('body')` to retrieve the JSON (request) content. The database server does not parse the JSON input automatically. It is up to the application to parse it.

- **'XML'** The result set is returned as XML. If the result set is already XML, no additional formatting is applied. Otherwise, it is automatically formatted as XML. As an alternative approach, a RAW service could return a select using the FOR XML RAW clause having set a valid Content-Type such as text/xml using `sa_set_http_header` procedure.

URL clause Determines whether URL paths are accepted and, if so, how they are processed. Specifying URL PATH has the same effect as URL.

- **OFF** Indicates that the service name in a URL request must not be followed by a path. OFF is the default setting. For example, the following form is disallowed due to the path elements `/aaa/bbb/cc`.

`http://host-name/service-name/aaa/bbb/cc`

Suppose that `CREATE SERVICE echo URL PATH OFF` was specified when creating the web service. A URL similar to `http://localhost/echo?id=1` produces the following values:

Function call	Result
<code>HTTP_VARIABLE('id')</code>	1
<code>HTTP_HEADER('@HttpQueryString')</code>	id=1

- **ON** Indicates that the service name in a URL request can be followed by a path. The path value is returned by querying a dedicated HTTP variable called URL. A service can be defined to explicitly provide the URL parameter or it may be retrieved using the `HTTP_VARIABLE` function. For example, the following form is allowed:

`http://host-name/service-name/aaa/bbb/cc`

Suppose that `CREATE SERVICE echo URL PATH ON` was specified when creating the web service. A URL similar to `http://localhost/echo/one/two?id=1` produces the following values:

Function call	Result
<code>HTTP_VARIABLE('id')</code>	1
<code>HTTP_VARIABLE('URL')</code>	one/two
<code>HTTP_HEADER('@HttpQueryString')</code>	id=1

- **ELEMENTS** Indicates that the service name in a URL request may be followed by a path. The path is obtained in segments by specifying a single parameter keyword URL1, URL2, and so on. Each parameter may be retrieved using the `HTTP_VARIABLE` or `NEXT_HTTP_VARIABLE` functions. These iterator functions can be used in applications where a variable number of path elements can be provided. For example, the following form is allowed:

`http://host-name/service-name/aaa/bbb/cc`

Suppose that `CREATE SERVICE echo URL PATH ELEMENTS` was specified when creating the web service. A URL similar to `http://localhost/echo/one/two?id=1` produces the following values:

Function call	Result
HTTP_VARIABLE('id')	1
HTTP_VARIABLE('URL1')	one
HTTP_VARIABLE('URL2')	two
HTTP_VARIABLE('URL3')	NULL
HTTP_HEADER('@HttpQueryString')	id=1

Up to 10 elements can be obtained. A NULL value is returned if the corresponding element is not supplied. In the above example, `HTTP_VARIABLE('URL3')` returns NULL because no corresponding element was supplied.

AUTHORIZATION clause Determines whether users must specify a user name and password through basic HTTP authorization when connecting to the service. The default value is ON. If authorization is OFF, the AS clause is required for all services and a user must be specified with the USER clause. All requests are run using that user's account and privileges. If AUTHORIZATION is ON, all users must provide a user name and password. Optionally, you can limit the users that are permitted to use the service by providing a user or group name with the USER clause. If the user name is NULL, all known users can access the service. The AUTHORIZATION clause allows your web services to use database authorization and privileges to control access to the data in your database.

When the authorization value is ON, an HTTP client connecting to a web service uses basic authentication (RFC 2617) that obfuscates the user and password information using base-64 encoding. Use the HTTPS protocol for increased security.

ENABLE and DISABLE clauses Determines whether the service is available for use. By default, when a service is created, it is enabled. When creating or altering a service, you may include an ENABLE or DISABLE clause. Disabling a service effectively takes the service off line. Later, it can be enabled using ALTER SERVICE with the ENABLE clause. An HTTP request made to a disabled service typically returns a `404 Not Found` HTTP status.

METHODS clause Specifies the HTTP methods that are supported by the service. Valid values are DEFAULT, POST, GET, HEAD, PUT, DELETE, and NONE. An asterisk (*) may be used as a short form to represent the POST, GET, and HEAD methods which are default request types for the RAW, HTML, and XML service types. Not all HTTP methods are valid for all the service types. The following table summarizes the valid HTTP methods that can be applied to each service type:

Method value	Applies to service	Description
DEFAULT	all	Use DEFAULT to reset the set of default HTTP methods for the given service type. It cannot be included in a list with other method values.
POST	RAW, HTML, JSON, XML	Enabled by default.
GET	RAW, HTML, JSON, XML	Enabled by default.
HEAD	RAW, HTML, JSON, XML	Enabled by default.
PUT	RAW, HTML, JSON, XML	Not enabled by default.
DELETE	RAW, HTML, JSON, XML	Not enabled by default.

Method value	Applies to service	Description
NONE	all	Use NONE to disable access to a service.
*	RAW, HTML, JSON, XML	Same as specifying 'POST,GET,HEAD'.

For example, you can use either of the following clauses to specify that a service supports all HTTP method types:

```
METHODS '*' PUT,DELETE'
METHODS 'POST,GET,HEAD,PUT,DELETE'
```

To reset the list of request types for any service type to its default, you can use the following clause:

```
METHODS 'DEFAULT'
```

SECURE clause Specifies whether the service should be accessible on a secure or non-secure listener. ON indicates that only HTTPS connections are accepted, and that connections received on the HTTP port are automatically redirected to the HTTPS port. OFF indicates that both HTTP and HTTPS connections are accepted, provided that the necessary ports are specified when starting the web server. The default value is OFF.

USER clause Specifies a database user, or group of users, with privileges to execute the web service request. A USER clause must be specified when the service is configured with AUTHORIZATION OFF and should be specified with AUTHORIZATION ON (the default). An HTTP request made to a service requiring authorization results in a [401 Authorization Required](#) HTTP response status. Based on this response, the web browser prompts for a user ID and password.

Caution

It is strongly recommended that you specify a USER clause when authorization is enabled (default). Otherwise, authorization is granted to all users.

The USER clause controls which database user accounts can be used to process service requests. Database access permissions are restricted to the privileges assigned to the user of the service.

statement Specifies a command, such as a stored procedure call, to invoke when the service is accessed.

An HTTP request to a non-DISH service with no *statement* specifies the SQL expression to execute within its URL. Although authorization is required, this capability should not be used in production systems because it makes the server vulnerable to SQL injections. When a statement is defined within the service, the specified SQL statement is the only statement that can be executed through the service.

In a typical web service application, you use *statement* to call a function or procedure. You can pass host variables as parameters to access client-supplied HTTP variables.

The following *statement* demonstrates a procedure call that passes two host variables to a procedure called AuthenticateUser. This call presumes that a web client supplies the user_name and user_password variables:

```
CALL AuthenticateUser ( :user_name, :user_password );
```

Remarks

The ALTER SERVICE statement modifies the attributes of a web service.

Privileges

You must be the owner of the service, or have the MANAGE ANY WEB SERVICE system privilege.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example demonstrates how to disable an existing web service using the ALTER SERVICE statement:

```
CREATE SERVICE WebServiceTable
  TYPE 'RAW'
  AUTHORIZATION OFF
  USER DBA
  AS SELECT *
  FROM SYS.SYSTAB;
```

```
ALTER SERVICE WebServiceTable DISABLE;
```

Related Information

[The database server as an HTTP web server](#)

The database server contains a built-in HTTP web server that allows you to provide online web services from a database.

[How to create and customize a root web service](#)

An HTTP client request that does not match any web service request is processed by the **root** web service if a **root** web service is defined.

[How to develop web service applications in an HTTP web server](#)

Customized web pages can be produced by the stored procedures that are called from web services. A very elementary web page example is shown below.

[Tutorial: Using a database server to access a SOAP/DISH service](#)

This tutorial illustrates how to create a SOAP server that converts a web client-supplied Fahrenheit temperature value to Celsius.

[CREATE SERVICE statement \[HTTP web service\]](#)

Creates a new HTTP web service.

[DROP SERVICE statement](#)

Drops a web service.

[sp_parse_json system procedure](#)

Returns a representation of JSON data using SQL data types.

[SYSWEBSERVICE system view](#)

Each row in the SYSWEBSERVICE system view holds a description of a web service. The underlying system table for this view is ISYSWEBSERVICE.

[-xs database server option](#)

Specifies server-side web services communications protocols.

[sa_set_http_header system procedure](#)

Permits a web service to set an HTTP response header.

[Identifiers](#)

Identifiers are the names of objects in the database, such as user IDs, tables, and columns.

[ROW constructor \[Composite\]](#)

Returns a sequence of (*field name data type*, ...) pairs named **fields**.

[ARRAY constructor \[Composite\]](#)

Returns elements of a specific data type.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

ALTER SERVICE statement [SOAP web service]

Alters an existing SOAP or DISH service over HTTP.

Syntax

- **SOAP over HTTP services**

```
ALTER SERVICE service-name
[ TYPE 'SOAP' ]
[ DATATYPE { ON | OFF | IN | OUT } ]
[ FORMAT { 'DNET' | 'CONCRETE' [ EXPLICIT { ON | OFF } ] | 'XML' | NULL } ]
[ common-attributes ]
[ AS statement ]
```

common-attributes :

```
[ AUTHORIZATION { ON | OFF } ]
[ ENABLE | DISABLE ]
[ METHODS 'method,...' ]
[ SECURE { ON | OFF } ]
[ USER { user-name | NULL } ]
```

method :

```
DEFAULT
| POST
| HEAD
| NONE
```

- **DISH services**

```
ALTER SERVICE service-name
[ TYPE 'DISH' ]
[ GROUP { group-name | NULL } ]
[ FORMAT { 'DNET' | 'CONCRETE' [ EXPLICIT { ON | OFF } ] | 'XML' | NULL } ]
[ common-attributes ]
```

common-attributes :

```
[ AUTHORIZATION { ON | OFF } ]
[ ENABLE | DISABLE ]
[ METHODS 'method,...' ]
[ SECURE { ON | OFF } ]
[ USER { user-name | NULL } ]
```

method :

```
DEFAULT
| POST
| GET
| HEAD
| NONE
| *
```

Parameters

The descriptions of the ALTER SERVICE clauses are identical to those of the CREATE SERVICE statement.

Parameters

service-name Web service names can be any sequence of alphanumeric characters or slash (/), hyphen (-), underscore (_), period (.), exclamation mark (!), tilde (~), asterisk (*), apostrophe ('), left parenthesis ((), or right parenthesis ()), except that the service name must not begin or end with a slash (/) or contain two or more consecutive slashes (for example, //).

Unlike other services, you cannot use a slash (/) anywhere in a DISH service name.

You can name your service root, but this name has a special function.

TYPE clause Identifies the type of the service where each service defines a specific response format. The type must be one of the listed service types. There is no default value.

- **'SOAP'** The result set is returned as an XML payload known as a SOAP envelope. The format of the data may be further refined using by the FORMAT clause. A request to a SOAP service must be a valid SOAP request, not just a general HTTP request. For more information about the SOAP standards, see [Simple Object Access Protocol \(SOAP\)](#).
- **'DISH'** A DISH service (Determine SOAP Handler) is a SOAP endpoint that references any SOAP service within its GROUP context. It also exposes the interfaces to its SOAP services by generating a WSDL (Web Services Description Language) for consumption by SOAP client toolkits.

GROUP clause A DISH service without a GROUP clause exposes all SOAP services defined within the database. By convention, the SOAP service name can be composed of a GROUP and a NAME element. The name is delimited from the group by the last slash character. For example, a SOAP service name defined as 'aaa/bbb/ccc' is 'ccc', and the group is 'aaa/bbb'. Delimiting a DISH service using this convention is invalid. Instead, a GROUP clause is applied to specify the group of SOAP services for which it is to be the SOAP endpoint.

Note Slashes are converted to underscores within the WSDL to produce valid XML. Use caution when using a DISH service that does not specify a GROUP clause such that it exposes all SOAP services that may contain slashes. Use caution when using groups with SOAP service names that contain underscores to avoid ambiguity.

DATATYPE clause Applies to SOAP services only. When DATATYPE OFF is specified, SOAP input parameters and response data are defined as XMLSchema string types. In most cases, true data types are preferred because it negates the need for the SOAP client to cast the data prior to computation. Parameter data types are exposed in the schema section of the WSDL generated by the DISH service. Output data types are represented as XML schema type attributes for each column of data.

The following values are permitted for the DATATYPE clause:

- **ON** Generates data typing of input parameters and result set responses.
- **OFF** All input parameters and response data are typed as XMLSchema string (default).
- **IN** Generates true data types for input parameters only. Response data types are XMLSchema string.
- **OUT** Generates true data types for responses only. Input parameters are typed as XMLSchema string.

FORMAT clause This clause specifies the output format when sending responses to SOAP client applications.

The SOAP service format is dictated by the associated DISH service format specification when it is not specified by the SOAP service. The default format is DNET.

SOAP requests should be directed to the DISH service (the SQL Anywhere SOAP endpoint) to leverage common formatting rules for a group of SOAP services (SOAP operations). A SOAP service FORMAT specification overrides that of a DISH service. The format specification of the DISH service is used when a SOAP service does not define a FORMAT clause. If no FORMAT is provided by either service then the default is 'DNET'.

The following formats are supported:

- o **'DNET'** The output is in a System.Data.DataSet compatible format for consumption by .NET client applications. (default)
- o **'CONCRETE'** This output format is used to support client SOAP toolkits that are capable of generating interfaces representing arrays of row and column objects but are not able to consume the DNET format. Java and .NET clients can easily consume this output format.

The specific format is exposed within the WSDL as an explicit dataset object or a SimpleDataset. Both dataset representations describe a data structure representing an array of rows with each row containing an array of columns. An explicit dataset object has the advantage of representing the actual shape of the result set by providing parameter names and datatypes for each column in the row. In contrast, the SimpleDataset exposes rows containing an unbounded number of columns of any type.

FORMAT 'CONCRETE' EXPLICIT ON requires that the Service statement calls a stored procedure which defines a RESULT clause. Having met this condition, the SOAP service will expose an explicit dataset whose name begins with the service name appended with Dataset.

If the condition is not met, a SimpleDataset is used.

- o **'XML'** The output is generated in an XMLSchema string format. The response is an XML document that requires further processing by the SOAP client to extract column data. This format is suitable for SOAP clients that cannot generate intermediate interface objects that represent arrays of rows and columns.
- o **NULL** A NULL type causes the SOAP or DISH service to use the default behavior. The format type of an existing service is overwritten when using the NULL type in an ALTER SERVICE statement.

AUTHORIZATION clause Determines whether users must specify a user name and password through basic HTTP authorization when connecting to the service. The default value is ON. If authorization is OFF, the AS clause is required for SOAP services, and a user must be specified with the USER clause. All requests are run using that user's account and privileges. If AUTHORIZATION is ON, all users must provide a user name and password. Optionally, you can limit the users that are permitted to use the service by providing a user or group name with the USER clause. If the user name is NULL, all known users can access the service. The AUTHORIZATION clause allows your web services to use database authorization and privileges to control access to the data in your database.

When the authorization value is ON, an HTTP client connecting to a web service uses basic authentication (RFC 2617) which obfuscates the user and password information using base-64 encoding. It is recommended that you use the HTTPS protocol for increased security.

ENABLE and DISABLE clauses Determines whether the service is available for use. By default, when a service is created, it is enabled. When creating or altering a service, you may include an ENABLE or DISABLE clause. Disabling a service effectively takes the service off line. Later, it can be enabled using ALTER SERVICE with the ENABLE clause. An HTTP request made to a disabled service typically returns a [404 Not Found](#) HTTP status.

METHODS clause Specifies the HTTP methods that are supported by the service. Valid values are DEFAULT, POST, GET, HEAD, and NONE. An asterisk (*) may be used as a short form to represent the POST, GET, and HEAD methods. The default method types for SOAP services are POST and HEAD. The default method types for DISH services are GET, POST, and HEAD. Not all HTTP methods are valid for all the service types. The following table summarizes the valid HTTP methods that can be applied to each service type:

Method value	Applies to service	Description
DEFAULT	both	Use DEFAULT to reset the set of default HTTP methods for the given service type. It cannot be included in a list with other method values.
POST	both	Enabled by default for SOAP.
GET	DISH only	Enabled by default for DISH.
HEAD	both	Enabled by default for SOAP and DISH.
NONE	both	Use NONE to disable access to a service. When applied to a SOAP service, the service cannot be directly accessed by a SOAP request. This enforces exclusive access to a SOAP operation through a DISH service SOAP endpoint. It is recommended that you specify METHODS 'NONE' for each SOAP service.
*	DISH only	Same as specifying 'POST,GET,HEAD' .

For example, you can use the following clause to specify that a service supports all SOAP over HTTP method types:

```
METHODS 'POST,HEAD'
```

To reset the list of request types for any service type to its default, you can use the following clause:

```
METHODS 'DEFAULT'
```


SECURE clause Specifies whether the service should be accessible on a secure or non-secure listener. ON indicates that only HTTPS connections are accepted, and that connections received on the HTTP port are automatically redirected to the HTTPS port. OFF indicates that both HTTP and HTTPS connections are accepted, provided that the necessary ports are specified when starting the web server. The default value is OFF.

USER clause Specifies a database user, or group of users, with privileges to execute the web service request. A USER clause must be specified when the service is configured with AUTHORIZATION OFF and should be specified with AUTHORIZATION ON (the default). An HTTP request made to a service requiring authorization results in a [401 Authorization Required](#) HTTP response status. Based on this response, the web browser prompts for a user ID and password.

Caution

It is strongly recommended that you specify a USER clause when authorization is enabled (the default). Otherwise, authorization is granted to all users.

The USER clause controls which database user accounts can be used to process service requests. Database access permissions are restricted to the privileges assigned to the user of the service.

statement Specifies a command, such as a stored procedure call, to invoke when the service is accessed.

A DISH service is the only service that must either define a null statement, or not define a statement. A SOAP service must define a statement. Any other SERVICE can have a NULL statement, but only if it is configured with AUTHORIZATION ON.

An HTTP request to a non-DISH service with no *statement* specifies the SQL expression to execute within its URL. Although authorization is required, this capability should not be used in production systems because it makes the server vulnerable to SQL injections. When a statement is defined within the service, the specified SQL statement is the only statement that can be executed through the service.

In a typical web service application, you use *statement* to call a function or procedure. You can pass host variables as parameters to access client-supplied HTTP variables.

The following *statement* demonstrates a procedure call that passes two host variables to a procedure named AuthenticateUser. This call presumes that a web client supplies the user_name and user_password variables:

```
CALL AuthenticateUser ( :user_name, :user_password );
```

Remarks

The ALTER SERVICE statement modifies the attributes of a web service.

Privileges

You must be the owner of the service, or have the MANAGE ANY WEB SERVICE system privilege.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example demonstrates how to disable an existing web service using the ALTER SERVICE statement:

```
CREATE SERVICE WebServiceTable
  TYPE 'SOAP'
  AUTHORIZATION OFF
  USER DBA
  AS SELECT *
    FROM SYS.SYSTAB;
```

```
ALTER SERVICE WebServiceTable DISABLE;
```

Related Information

[The database server as an HTTP web server](#)

The database server contains a built-in HTTP web server that allows you to provide online web services from a database.

[How to create and customize a root web service](#)

An HTTP client request that does not match any web service request is processed by the **root** web service if a **root** web service is defined.

[Tutorial: Using a database server to access a SOAP/DISH service](#)

This tutorial illustrates how to create a SOAP server that converts a web client-supplied Fahrenheit temperature value to Celsius.

[CREATE SERVICE statement \[SOAP web service\]](#)

Creates a new SOAP over HTTP or DISH service.

[DROP SERVICE statement](#)

Drops a web service.

[SYSWEBSERVICE system view](#)

Each row in the SYSWEBSERVICE system view holds a description of a web service. The underlying system table for this view is ISYSWEBSERVICE.

[-xs database server option](#)

Specifies server-side web services communications protocols.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

ALTER SPATIAL REFERENCE SYSTEM statement

Changes the settings of an existing spatial reference system. See the Remarks section for considerations before altering a spatial reference system.

Syntax

ALTER SPATIAL REFERENCE SYSTEM

srs-name

[*srs-attribute* [*srs-attribute* ...]]

srs-name : *string*

srs-attribute :

SRID *srs-id*

| **DEFINITION** { *definition-string* | **NULL** }

| **ORGANIZATION** { *organization-name* **IDENTIFIED BY** *organization-srs-id* | **NULL** }

| **TRANSFORM DEFINITION** { *transform-definition-string* | **NULL** }

| **LINEAR UNIT OF MEASURE** *linear-unit-name*

| **ANGULAR UNIT OF MEASURE** { *angular-unit-name* | **NULL** }

| **TYPE** { **ROUND EARTH** | **PLANAR** }

| **COORDINATE** *coordinate-name* { **UNBOUNDED** | **BETWEEN** *low-number* **AND** *high-number* }

| **ELLIPSOID SEMI MAJOR AXIS** *semi-major-axis-length* { **SEMI MINOR AXIS** *semi-minor-axis-length* | **INVERSE FLATTENING** *inverse-flattening-ratio* }

| **SNAP TO GRID** { *grid-size* | **DEFAULT** }

| **TOLERANCE** { *tolerance-distance* | **DEFAULT** }

| **AXIS ORDER** *axis-order*

| **POLYGON FORMAT** *polygon-format*

| **STORAGE FORMAT** *storage-format*

srs-id : *integer*

semi-major-axis-length : *number*

semi-minor-axis-length : *number*

inverse-flattening-ratio : *number*

grid-size : *DOUBLE* : usually between 0 and 1

tolerance-distance : *number*

axis-order : { 'x/y/z/m' | 'long/lat/z/m' | 'lat/long/z/m' }

polygon-format : { 'CounterClockWise' | 'Clockwise' | 'EvenOdd' }

storage-format : { 'Internal' | 'Original' | 'Mixed' }

Parameters

Complete definitions for each of the clauses is provided in the CREATE SPATIAL REFERENCE SYSTEM statement.

IDENTIFIED BY clause Use this clause to change the SRID number for the spatial reference system.

DEFINITION clause Use this clause to set, or override, default coordinate system settings.

ORGANIZATION clause Use this clause to specify information about the organization that created the spatial reference system that the spatial reference system is based on.

TRANSFORM DEFINITION clause Use this clause to specify a description of the transform to use for the spatial reference system. Currently, only the PROJ.4 transform is supported.

The transform definition is used by the ST_Transform method when transforming data between spatial reference systems. Some transforms may still be possible even if there is no *transform-definition-string* defined.

COORDINATE clause Use this clause to specify the bounds on the spatial reference system's dimensions. *coordinate-name* is the name of the coordinate system used by the spatial reference system. For non-geographic types *coordinate-name* can be x, y, or m. For geographic types, *coordinate-name* can be LATITUDE, LONGITUDE, z, or m.

LINEAR UNIT OF MEASURE clause Use this clause to specify the linear unit of measure for the spatial reference system. The value you specify must match a linear unit of measure defined in the ST_UNITS_OF_MEASURE consolidated view.

ANGULAR UNIT OF MEASURE clause Use this clause to specify the angular unit of measure for the spatial reference system. The value you specify must match an angular unit of measure defined in the ST_UNITS_OF_MEASURE consolidated view.

TYPE clause Use the TYPE clause to control how the spatial reference system interprets lines between points. For geographic spatial reference systems, the TYPE clause can specify either ROUND EARTH (the default) or PLANAR. For non-geographic spatial reference systems, the type must be PLANAR.

ELLIPSOID clause Use the ellipsoid clause to specify the values to use for representing the Earth as an ellipsoid for spatial reference systems of type ROUND EARTH. If the DEFINITION clause is present, it can specify ellipsoid definition. If the ELLIPSOID clause is specified, it overrides this default ellipsoid.

SNAP TO GRID clause For flat-Earth (planar) spatial reference systems, use the SNAP TO GRID clause to define the size of the grid the database server uses when performing calculations. Specify SNAP TO GRID DEFAULT to set the grid size to the default that the database server would use.

For round-Earth spatial reference systems, SNAP TO GRID must be set to 0.

TOLERANCE clause For flat-Earth (planar) spatial reference systems, use the TOLERANCE clause to specify the precision to use when comparing points.

For round-Earth spatial reference systems, TOLERANCE must be set to 0.

POLYGON FORMAT clause Use the POLYGON FORMAT clause to change the polygon interpretation. The following values are supported:

- 'CounterClockwise'
- 'Clockwise'
- 'EvenOdd'

The default polygon format is 'EvenOdd'.

STORAGE FORMAT clause Use the STORAGE FORMAT clause to control what is stored when spatial data is loaded into the database. Possible values are:

- o **'Internal'** The database server stores only the normalized representation. Specify this when the original input characteristics do not need to be reproduced. This is the default for planar spatial reference systems (TYPE PLANAR).

Note If you are using MobiLink to synchronize your spatial data, you should specify **Mixed** instead. MobiLink tests for equality during synchronization, which requires the data in its original format.

- o **'Original'** The database server stores only the original representation. The original input characteristics can be reproduced, but all operations on the stored values must repeat normalization steps, possibly slowing down operations on the data.
- o **'Mixed'** The database server stores the internal version and, if the internal version is different from the original version, the original version as well. By storing both versions, the original representation characteristics can be reproduced and operations on stored values do not need to repeat normalization steps. However, storage requirements may increase significantly because potentially two representations are being stored for each geometry.

Mixed is the default format for round-Earth spatial reference systems (TYPE ROUND EARTH).

Remarks

You cannot alter a spatial reference system if there is existing data that references it. For example, if you have a column declared as ST_Point(SRID=8743), you cannot alter the spatial reference system with SRID 8743. This is because many spatial reference system attributes, such as storage format, impact the storage format of the data. If you have data that references the SRID, create a new spatial reference system and transform the data to the new SRID.

If you use this statement in a procedure, do not specify the password (IDENTIFIED BY clause) as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

Privileges

You must be the owner of the spatial reference system, or have one of the following privileges:

- ALTER privilege on the spatial reference system
- MANAGE ANY SPATIAL OBJECT system privilege
- ALTER ANY OBJECT system privilege

Side effects

None

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example changes the polygon format of a fictitious spatial reference system named

mySpatialRef to EvenOdd.

```
ALTER SPATIAL REFERENCE SYSTEM mySpatialRef  
POLYGON FORMAT 'EvenOdd';
```

Related Information

[Spatial data](#)

Spatial data is data that describes the position, shape, and orientation of objects in a defined space.

[CREATE SPATIAL REFERENCE SYSTEM statement](#)

Creates or replaces a spatial reference system.

[DROP SPATIAL REFERENCE SYSTEM statement](#)

Drops a spatial reference system.

[ST_UNITS_OF_MEASURE consolidated view](#)

Each row of the ST_UNITS_OF_MEASURE system view describes a unit of measure defined in the database. This view offers more information than the SYSUNITOFMEASURE system view.

[ST_Transform method](#)

Creates a copy of the geometry value transformed into the specified spatial reference system.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

ALTER STATISTICS statement

Controls whether statistics are automatically updated on a column, or columns, in a table.

Syntax

```
ALTER STATISTICS  
[ ON ] table [ ( column1 [ , column2 ... ] ) ]  
AUTO UPDATE { ENABLE | DISABLE }
```

Parameters

ON The word ON is optional. Including it has no impact on the execution of the statement.

AUTO UPDATE clause Specify whether to enable or disable automatic updating of statistics for the column(s).

Remarks

During normal execution of queries, DML statements, and LOAD TABLE statements, the database server automatically maintains column statistics for use by the optimizer. The benefit of maintaining statistics for some columns may not outweigh the overhead necessary to generate them. For example, if a column is not queried often, or if it is subject to periodic mass changes that are eventually rolled back, there is little value in continually updating its statistics. Use the ALTER STATISTICS statement to suppress the automatic updating of statistics for these types of columns.

When automatic updating is disabled, you can still update the statistics for the column using the CREATE STATISTICS and DROP STATISTICS statements. However, you should only update them if it has been determined that it would have a positive impact on performance. Normally, column statistics should not be disabled.

Privileges

You must be the table owner, or have one of the following privileges:

- **MANAGE ANY STATISTICS** system privilege
- **ALTER ANY OBJECT** system privilege

Side effects

If automatic updating has been disabled, the statistics may become out of date. Re-enabling does not immediately bring them up to date. Execute the CREATE STATISTICS statement to recreate them, if necessary.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example disables the automatic updating of statistics on the Street column in the Customers table:

```
ALTER STATISTICS GROUP0.Customers ( Street ) AUTO UPDATE DISABLE;
```

Related Information

[CREATE STATISTICS statement](#)

Recreates the column statistics used by the optimizer, and stores them in the ISYSCOLSTAT system table.

[DROP STATISTICS statement](#)

Erases all column statistics on the specified columns.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

ALTER SYNCHRONIZATION PROFILE statement [MobiLink]

Changes a SQL Anywhere synchronization profile. Synchronization profiles are named collections of synchronization options that can be used to control synchronization.

Syntax

```
ALTER SYNCHRONIZATION PROFILE name  
MERGE string
```

Parameters

name The name of the synchronization profile to alter.

MERGE clause Use this clause to change existing, or add new, options to a synchronization profile.

string A string of one or more synchronization option value pairs, separated by semicolons. For example, 'option1=value1;option2=value2'.

Remarks

Synchronization profiles define how a SQL Anywhere database synchronizes with the MobiLink server.

When MERGE is used in the ALTER SYNCHRONIZATION PROFILE statement, options specified in the string are added to those already in the synchronization profile. If an option in the string already exists in profile, then the value from the string replaces the value already stored in the profile.

For example, executing the following statements leaves the profile *myProfile* with the value *subscription=s2;verbosity=high;uploadonly=on*.

```
CREATE SYNCHRONIZATION PROFILE myProfile 'subscription=p1;verbosity=high';  
ALTER SYNCHRONIZATION PROFILE myProfile MERGE 'subscription=p2;uploadonly=on';
```

When setting extended options, use the following syntax:

```
ALTER SYNCHRONIZATION PROFILE myprofile MERGE 's=mysub;e={ctp=tcip;  
adr=' 'host=localhost;port=2439' '}'
```

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Related Information

[CREATE SYNCHRONIZATION PROFILE statement \[MobiLink\]](#)

Creates a SQL Anywhere synchronization profile.

[DROP SYNCHRONIZATION PROFILE statement \[MobiLink\]](#)

Deletes a SQL Anywhere synchronization profile.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

ALTER SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]

Alters the properties of a synchronization subscription in a SQL Anywhere remote database.

Syntax

ALTER SYNCHRONIZATION SUBSCRIPTION

```
{ subscription-name | TO publication-name [ FOR ml-username, ... ] } { alter-clause ... }
```

alter-clause :

```
RENAME new-subscription-name  
| TYPE network-protocol  
| ADDRESS protocol-options  
| ADD OPTION option=value, ...  
| ALTER OPTION option=value, ...  
| DELETE { ALL OPTION | OPTION option, ... }  
| SET SCRIPT VERSION=script-version
```

subscription-name : *identifier*

publication-name : *identifier*

ml-username : *identifier*

new-subscription-name : *identifier*

network-protocol : **http** | **https** | **tls** | **tcip** | **NULL**

protocol-options : *string* | **NULL**

value : *string* | *integer*

option : *identifier*

script-version : *string* | **NULL**

Parameters

TO clause This clause specifies the name of a publication.

If the TO clause is used without a FOR clause, you cannot use the RENAME or SET SCRIPT VERSION clauses.

FOR clause This clause specifies one or more MobiLink user names.

Omit the FOR clause to set the protocol type, protocol options, and extended options for a publication.

If the TO clause is used without a FOR clause, you cannot use the RENAME or SET SCRIPT VERSION clauses.

RENAME clause This clause specifies a new name for the subscription.

If the TO clause is used without a FOR clause, you cannot use the RENAME clause.

TYPE clause This clause specifies the network protocol to use for synchronization. The default protocol is tcpip.

ADDRESS clause This clause specifies network protocol options, including the location of the MobiLink server.

ADD OPTION, ALTER OPTION, DELETE OPTION, and DELETE ALL OPTION clauses

These clauses allow you to add, alter, delete, or delete all extended options. You can specify only one option in each clause. No option is specified for Delete All.

The values for each option cannot contain the characters " = " or " , " or " ; ".

SET SCRIPT VERSION clause This clause specifies the script version to use during synchronization. You can alter the script version without making a schema change.

If the TO clause is used without a FOR clause, you cannot use the SET SCRIPT VERSION clause.

Remarks

The *network-protocol*, *protocol-options*, and *options* can be set in several places.

This statement causes options and other information to be stored in the SQL Anywhere ISYSSYNC system table. Depending on the privileges a user has, they may be able to view the information, which could include passwords and encryption certificates. To avoid this potential security issue, you can specify the information on the dbmlsync command line.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example changes the address of the MobiLink server for the sales subscription:

```
ALTER SYNCHRONIZATION SUBSCRIPTION sales
TYPE TCP/IP
ADDRESS 'host=10.11.12.132;port=2439';
```

Related Information

[Synchronization subscription creation](#)

After creating MobiLink users and publications, you must subscribe at least one MobiLink user to one or more pre-existing publications. You do this by creating synchronization subscriptions.

[MobiLink SQL Anywhere client extended options](#)

Extended options can be specified on the dbmlsync command line using the -e or -eu options, or they can be stored in the database.

[How dbmlsync resolves conflicting options](#)

Dbmlsync combines options stored in the database with those specified on the command line. If conflicting options are specified, dbmlsync resolves them as follows. In the following list, options

specified by methods occurring earlier in the list take precedence over those occurring later in the list.

[Script versions](#)

Scripts are organized into groups called **script versions**. By specifying a particular script version, MobiLink clients can select which set of synchronization scripts are used to process the upload and prepare the download.

[CREATE PUBLICATION statement \[MobiLink\] \[SQL Remote\]](#)

Creates a publication. In MobiLink, a publication identifies synchronized data in a SQL Anywhere remote database. In SQL Remote, publications identify replicated data in both consolidated and remote databases.

[DROP PUBLICATION statement \[MobiLink\] \[SQL Remote\]](#)

Drops a publication.

[CREATE SYNCHRONIZATION SUBSCRIPTION statement \[MobiLink\]](#)

Creates a subscription in a SQL Anywhere remote database between a MobiLink user and a publication.

[SYSSYNC system view](#)

The SYSSYNC system view contains information relating to synchronization.

[MobiLink client network protocol options](#)

MobiLink client utilities use the following MobiLink client network protocol options when connecting to the MobiLink server.

[CommunicationType \(ctp\) extended option](#)

Specifies the type of network protocol to use for connecting to the MobiLink server.

[MobiLink SQL Anywhere client utility \(dbmlsync\) syntax](#)

Use the dbmlsync utility to synchronize SQL Anywhere remote databases with a consolidated database. To run dbmlsync you must have the SYS_RUN_REPLICATION_ROLE system privilege.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

ALTER SYNCHRONIZATION USER statement [MobiLink]

Alters the properties of a MobiLink user in a SQL Anywhere remote database.

Syntax

```
ALTER SYNCHRONIZATION USER ml-username  
[ TYPE network-protocol ]  
[ ADDRESS protocol-options ]  
[ ADD OPTION option=value, ... ]  
[ ALTER OPTION option=value, ... ]  
[ DELETE { ALL OPTION | OPTION option } ]
```

ml-username : *identifier*

network-protocol : **http** | **https** | **tls** | **tcpip** | **NULL**

protocol-options : *string* | **NULL**

value : *string* | *integer*

Parameters

TYPE clause This clause specifies the network protocol to use for synchronization. The default protocol is tcpip.

ADDRESS clause This clause specifies *protocol-options* in the form *keyword=value*, separated by semicolons. Which settings you supply depends on the communication protocol you are using (TCP/IP, TLS, HTTP, or HTTPS).

ADD OPTION, ALTER OPTION, DELETE OPTION, and DELETE ALL OPTION clauses

These clauses allow you to add, modify, delete, or delete all extended options. You may specify only one option in each clause. No option is specified for Delete All.

Remarks

The *network-protocol*, *protocol-options*, and *options* can be set in several places.

This statement causes options and other information to be stored in the ISYSSYNC system table. Depending on the privileges a user has, passwords and encryption certificates can be visible. To avoid this potential security issue, you can specify the information on the dbmlsync command line.

Requires exclusive access to all tables referred to in the publication.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Related Information

[MobiLink users in a synchronization system](#)

A **MobiLink user**, also called a **synchronization user**, is the name you use to authenticate when you connect to the MobiLink server.

[MobiLink SQL Anywhere client extended options](#)

Extended options can be specified on the dbmlsync command line using the -e or -eu options, or they can be stored in the database.

[How dbmlsync resolves conflicting options](#)

Dbmlsync combines options stored in the database with those specified on the command line. If conflicting options are specified, dbmlsync resolves them as follows. In the following list, options specified by methods occurring earlier in the list take precedence over those occurring later in the list.

[MobiLink SQL Anywhere client utility \(dbmlsync\) syntax](#)

Use the dbmlsync utility to synchronize SQL Anywhere remote databases with a consolidated database. To run dbmlsync you must have the SYS_RUN_REPLICATION_ROLE system privilege.

[CREATE SYNCHRONIZATION USER statement \[MobiLink\]](#)

Creates a MobiLink user in a SQL Anywhere remote database.

[CommunicationType \(ctp\) extended option](#)

Specifies the type of network protocol to use for connecting to the MobiLink server.

[DROP SYNCHRONIZATION USER statement \[MobiLink\]](#)

Drops one or more synchronization users from a SQL Anywhere remote database.

[SYSSYNC system view](#)

The SYSSYNC system view contains information relating to synchronization.

[MobiLink client network protocol options](#)

MobiLink client utilities use the following MobiLink client network protocol options when connecting to the MobiLink server.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

ALTER TABLE statement

Modifies a table definition or disables dependent views.

Syntax

- **Altering an existing table**

```
ALTER TABLE [owner.]table-name { alter-clause, ... }
```

alter-clause :

ADD *create-clause*

| **ALTER** *column-name* *column-alteration*

| **ALTER** [**CONSTRAINT** *constraint-name*] **CHECK** (*condition*)

| **DROP** *drop-object*

| **RENAME** *rename-object*

| *table-alteration*

create-clause :

column-name [**AS**] *column-data-type* [*new-column-attribute* ...]

| *table-constraint*

| **PCTFREE** *integer*

column-alteration :

{ *column-data-type* | *alterable-column-attribute* } [*alterable-column-attribute* ...]

| **SET COMPUTE** (*compute-expression*)

| [**SET**] **DEFAULT** *default-value*

| **ADD** [*constraint-name*] **CHECK** (*condition*)

| **DROP** { **DEFAULT** | **COMPUTE** | **CHECK** | **CONSTRAINT** *constraint-name* }

drop-object :

column-name

| **CHECK**

| **CONSTRAINT** *constraint-name*

| **UNIQUE** [**CLUSTERED**] (*index-columns-list*)

| **FOREIGN KEY** *fkey-name*

| **PRIMARY KEY**

rename-object :

new-table-name

| *column-name* **TO** *new-column-name*

| **CONSTRAINT** *constraint-name* **TO** *new-constraint-name*

table-alteration :

PCTFREE DEFAULT

| [**NOT**] **ENCRYPTED**

new-column-attribute :

[**NOT**] **NULL**
 [**SET**] **DEFAULT** *default-value*
COMPRESSED
INLINE { *inline-length* | **USE DEFAULT** }
PREFIX { *prefix-length* | **USE DEFAULT** }
 [**NO**] **INDEX**
IDENTITY
COMPUTE (*expression*)
column-constraint

table-constraint :

[**CONSTRAINT** *constraint-name*] {
 CHECK (*condition*)
 | **UNIQUE** [**CLUSTERED** | **NONCLUSTERED**] (*column-name* [**ASC** | **DESC**], ...)
 | **PRIMARY KEY** [**CLUSTERED** | **NONCLUSTERED**] (*column-name* [**ASC** | **DESC**],
 ...)
 | *foreign-key*
 }

column-constraint :

[**CONSTRAINT** *constraint-name*] {
 CHECK (*condition*)
 | **UNIQUE** [**CLUSTERED** | **NONCLUSTERED**] [**ASC** | **DESC**]
 | **PRIMARY KEY** [**CLUSTERED** | **NONCLUSTERED**] [**ASC** | **DESC**]
 | **REFERENCES** *table-name* [(*column-name*)]
 [**MATCH** [**UNIQUE**] { **SIMPLE** | **FULL** }]
 [*actions*] [**CLUSTERED** | **NONCLUSTERED**]
 | **NOT NULL**
 }

alterable-column-attribute :

[**NOT**] **NULL**
DEFAULT *default-value*
 [**CONSTRAINT** *constraint-name*] **CHECK** { **NULL** | (*condition*) }
 [**NOT**] **COMPRESSED**
INLINE { *inline-length* | **USE DEFAULT** }
PREFIX { *prefix-length* | **USE DEFAULT** }
 [**NO**] **INDEX**

default-value :

special-value
 | *string*
 | *global variable*
 | [-] *number*
 | (*constant-expression*)
 | (*sequence-expression*)
 | *built-in-function*(*constant-expression*)
AUTOINCREMENT
GLOBAL AUTOINCREMENT [(*partition-size*)]

*special-value :***CURRENT DATABASE**| **CURRENT DATE**| **CURRENT TIME**| [**CURRENT**] **TIMESTAMP**| **CURRENT PUBLISHER**| **CURRENT REMOTE USER**| [**CURRENT**] **USER**| [**CURRENT**] **UTC TIMESTAMP**| **LAST USER**| **NULL***foreign-key :*[**NOT NULL**] **FOREIGN KEY** [*role-name*][(*column-name* [**ASC** | **DESC**], ...)**REFERENCES** *table-name*[(*pkey-column-list*)][**MATCH** [**UNIQUE**] { **SIMPLE** | **FULL** }][*actions*] [**CHECK ON COMMIT**] [**CLUSTERED**][**FOR OLAP WORKLOAD**]*actions :*[**ON UPDATE** *action*] [**ON DELETE** *action*]*action :***CASCADE** | **SET NULL** | **SET DEFAULT** | **RESTRICT**

- **Disabling view dependencies**

```
ALTER TABLE [owner.]table-name {
  DISABLE VIEW DEPENDENCIES
}
```

- **Altering a table owner**

```
ALTER TABLE [owner.]table-name ALTER OWNER TO owner
[ { PRESERVE | DROP } PRIVILEGES ]
[ { PRESERVE | DROP } FOREIGN KEYS ]
```

Parameters

Adding clauses There are several clauses you can use for adding columns or table constraints to a table:

ADD *column-name* [**AS] *column-data-type* [*new-column-attribute* ...] clause**

Use this clause to add a new column to the table, specifying the data type and attributes for the column. Define the data type explicitly, or specify the %TYPE attribute to set the data type to the data type of a column in another table or view. When you specify the %TYPE attribute, other attributes on the object referenced in the %TYPE specification, such as default values, constraints, and whether NULLs are allowed, are not part of the definition that is inherited and must be specified separately.

NULL and NOT NULL clauses Use this clause to specify whether to allow NULLs in the

column. By default, new columns allow NULL values. BIT type columns automatically have the NOT NULL constraint applied when they are created, but you can declare a BIT type column to be nullable.

DEFAULT clause If a DEFAULT value is specified, it is used as the value for the column in any INSERT statement that does not specify a value for the column. If no DEFAULT value is specified, it is equivalent to DEFAULT NULL.

Following is a list of possible values for DEFAULT:

- **constant-expression** Constant expressions that do not reference database objects are allowed in a DEFAULT clause, so 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.
- **global-variable** A global variable.
- **sequence-expression** Set DEFAULT to the current value or next value from a sequence in the database.
- **string** A string value.
- **AUTOINCREMENT** When using AUTOINCREMENT, the column must be one of the integer data types, or an exact numeric type.

On inserts into the table, if a value is not specified for the AUTOINCREMENT column, a unique value larger than any other value in the column is generated. If an INSERT specifies a value for the column that is larger than the current maximum value for the column, that value is inserted and then 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 an explicit insert of a column value less than the maximum, subsequent rows without explicit assignment are still automatically incremented with a value of one greater than the previous maximum.

You can find the most recently inserted value of the column by inspecting the @@identity global variable.

AUTOINCREMENT values are maintained as signed 64-bit integers, corresponding to the data type of the max_identity column in the SYSTABCOL system view. When the next value to be generated exceeds the maximum value that can be stored in the column to which the AUTOINCREMENT is assigned, NULL is returned. If the column has been declared to not allow NULLs, as is true for primary key columns, a SQL error is generated.

The next value to use for a column can be reset using the sa_reset_identity procedure.

- **GLOBAL AUTOINCREMENT** This default is intended for use when multiple databases are used in a MobiLink synchronization environment or SQL Remote replication.

This option is similar to AUTOINCREMENT, except that the domain is partitioned. Each partition contains the same number of values. You assign each copy of the database a unique global database identification number. The database server supplies default values in a database only from the partition uniquely identified by that database's number.

The partition size can be specified in parentheses immediately following the AUTOINCREMENT keyword. The partition size can be any positive integer, although the

partition size is generally chosen so that the supply of numbers within any one partition will rarely, if ever, be exhausted.

If the column is of type BIGINT or UNSIGNED BIGINT, the default partition size is $2^{32} = 4294967296$; for columns of all other types, the default partition size is $2^{16} = 65536$. Since these defaults may be inappropriate, especially if your column is not of type INT or BIGINT, it is best to specify the partition size explicitly.

When using this default, the value of the public option `global_database_id` in each database must be set to a unique, non-negative integer. This value uniquely identifies the database and indicates from which partition default values are to be assigned. The range of allowed values is $np + 1$ to $p(n + 1)$, where n is the value of the public option `global_database_id` and p is the partition size. For example, if you define the partition size to be 1000 and set `global_database_id` to 3, then the range is from 3001 to 4000.

If the previous value is less than $p(n + 1)$, the next default value is one greater than the previous largest value in the column. If the column contains no values, the first default value is $np + 1$. Default column values are not affected by values in the column outside the current partition; that is, by numbers less than $np + 1$ or greater than $p(n + 1)$. Such values may be present if they have been replicated from another database via MobiLink or SQL Remote.

You can find the most recently inserted value of the column by inspecting the `@@identity` global variable.

GLOBAL AUTOINCREMENT values are maintained as signed 64-bit integers, corresponding to the data type of the `max_identity` column in the SYSTABCOL system view. When the supply of values within the partition has been exhausted, NULL is returned. If the column has been declared to not allow NULLs, as is true for primary key columns, a SQL error is generated. In this case, a new value of `global_database_id` should be assigned to the database to allow default values to be chosen from another partition. To detect that the supply of unused values is low and handle this condition, create an event of type GlobalAutoincrement.

Because the public option `global_database_id` cannot be set to a negative value, the values chosen are always positive. The maximum identification number is restricted only by the column data type and the partition size.

If the public option `global_database_id` is set to the default value of 2147483647, a NULL value is inserted into the column. If NULL values are not permitted, attempting to insert the row causes an error.

The next value to use for a column can be reset using the `sa_reset_identity` procedure.

- **special-value** You use one of several special values in the DEFAULT clause (for example, CURRENT DATE) including, but not limited to, the following special values.
 - **[CURRENT] TIMESTAMP** Provides a way of indicating when each row in the table was last modified. When a column is declared with DEFAULT TIMESTAMP, a default value is provided for inserts, and the value is updated with the current date and time of day whenever the row is updated.

To provide a default value on insert, but not update the column whenever the row is updated, use DEFAULT CURRENT TIMESTAMP instead of DEFAULT TIMESTAMP.

Columns declared with DEFAULT TIMESTAMP contain unique values, so that applications can detect near-simultaneous updates to the same row. If the current

TIMESTAMP value is the same as the last value, it is incremented by the value of the `default_timestamp_increment` option.

You can automatically truncate TIMESTAMP values based on the `default_timestamp_increment` option. This is useful for maintaining compatibility with other database software that records less precise timestamp values.

The global variable `@@dbts` returns a TIMESTAMP value representing the last value generated for a column using DEFAULT TIMESTAMP.

- **[CURRENT] UTC TIMESTAMP** Provides a way of indicating when each row in the table was last modified. When a column is declared with DEFAULT UTC TIMESTAMP, a default value is provided for inserts, and the value is updated with the current Coordinated Universal Time (UTC) whenever the row is updated.

To provide a default value on insert, but not update the column whenever the row is updated, use DEFAULT CURRENT UTC TIMESTAMP instead of DEFAULT UTC TIMESTAMP.

The behavior of this default is the same as TIMESTAMP and CURRENT TIMESTAMP except that the date and time of day is in Coordinated Universal Time (UTC).

- **LAST USER** LAST USER is the user ID of the user who last modified the row.

LAST USER can be used as a default value in columns with character data types.

On INSERT, this default has the same effect as CURRENT USER.

On UPDATE, if a column with a default of LAST USER is not explicitly modified, it is changed to the name of the current user.

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

column-constraint clause Use this clause to add a constraint to the column.

Note With the exception of CHECK constraints, when a new constraint is added, the database server validates existing values to confirm that they satisfy the constraint. CHECK constraints are enforced only for operations that occur after the table alteration is complete.

Possible column constraints include:

- **CHECK clause** This 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 M or F.

If you need to create a CHECK constraint that involves a relationship between two or more columns in the table (for example, column A must be less than column B), define a table constraint instead.

- **UNIQUE clause** Use this subclause to specify that values in the column must be unique, and whether to create a clustered or nonclustered index.
- **PRIMARY KEY clause** Use this subclause to make the column a primary key, and specify whether to use a clustered index.
- **REFERENCES clause** Use this subclause to add or alter a reference to another table, to specify how matches are handled, and to specify whether to use a clustered

index.

- **MATCH clause** Use this subclause to control what is considered a match when using a multi-column foreign key. It also allows you to specify uniqueness for the key, thereby eliminating the need to declare uniqueness separately.
- **NULL and NOT NULL clauses** Use this clause to specify whether to allow NULL values in the column. By default, NULLs are allowed.

COMPRESSED clause Use this clause to compress the column.

INLINE and PREFIX clauses The **INLINE** clause specifies the maximum BLOB size, in bytes, to store within the row. BLOBs smaller than or equal to the value specified by the **INLINE** clause are stored within the row. BLOBs that exceed the value specified by the **INLINE** clause are stored outside the row in table extension pages. Also, a copy of some bytes from the beginning of the BLOB may be kept in the row when a BLOB is larger than the **INLINE** value. Use the **PREFIX** clause to specify how many bytes are kept in the row. The **PREFIX** clause can improve the performance of requests that need the prefix bytes of a BLOB to determine if a row is accepted or rejected.

The prefix data for a compressed column is stored uncompressed, so if all the data required to satisfy a request is stored in the prefix, no decompression is necessary.

If neither **INLINE** nor **PREFIX** is specified, or if **USE DEFAULT** is specified, default values are applied as follows:

- For character data type columns, such as **CHAR**, **NCHAR**, and **LONG VARCHAR**, the default value of **INLINE** is 256, and the default value of **PREFIX** is 8.
- For binary data type columns, such as **BINARY**, **LONG BINARY**, **VARBINARY**, **BIT**, **VARBIT**, **LONG VARBIT**, **BIT VARYING**, and **UUID**, the default value of **INLINE** is 256, and the default value of **PREFIX** is 0.

Note It is strongly recommended that you use the default values unless there are specific circumstances that require a different setting. The default values have been chosen to balance performance and disk space requirements. For example, if you set **INLINE** to a large value, and all the BLOBs are stored inline, row processing performance may degrade. If you set **PREFIX** too high, you increase the amount of disk space required to store BLOBs since the prefix data is a duplicate of a portion of the BLOB.

If only one of the values is specified, the other value is automatically set to the largest amount that does not conflict with the specified value. Neither the **INLINE** nor **PREFIX** value can exceed the database page size. Also, there is a small amount of overhead reserved in a table page that cannot be used to store row data. Therefore, specifying an **INLINE** value approximate to the database page size can result in a slightly smaller number of bytes being stored inline.

INDEX and NO INDEX clauses When storing BLOBs (character or binary types only), specify **INDEX** to create BLOB indexes on inserted values that exceed the internal BLOB size threshold (approximately eight database pages). This is the default behavior.

BLOB indexes can improve performance when random access searches within the BLOBs are required. However, for some types of BLOB values, such as images and multimedia files that will never require random-access, performance can improve if BLOB indexing is turned off. To turn off BLOB indexing for a column, specify **NO INDEX**.

Note A BLOB index is not the same as a table index. A table index is created to index values in one or more columns.

IDENTITY clause The IDENTITY clause is a Transact-SQL-compatible alternative to using DEFAULT AUTOINCREMENT. A column defined with IDENTITY is implemented as DEFAULT AUTOINCREMENT.

COMPUTE clause When a column is created using a COMPUTE clause, its value in any row is the value of the supplied expression. Columns created with this constraint are read-only columns for applications: the value is changed by the database server whenever the row is modified. The COMPUTE expression should not return a non-deterministic value. For example, it should not include a special value such as CURRENT_TIMESTAMP, or a non-deterministic function. If a COMPUTE expression returns a non-deterministic value, then it cannot be used to match an expression in a query.

The COMPUTE clause is ignored for remote tables.

Any UPDATE statement that attempts to change the value of a computed column fires any triggers associated with the column.

ADD table-constraint clause Use this clause to add a table constraint. Table constraints place limits on what data columns in the table can hold. When adding or altering table constraints, the optional constraint name allows you to modify or drop individual constraints. Following is a list of the table constraints you can add.

- **UNIQUE** Use this subclause to specify that values in the columns specified in *column-list* must be unique, and, optionally, whether to use a clustered index.
- **PRIMARY KEY** Use this subclause to add or alter the primary key for the table, and specify whether to use a clustered index. The table must not already have a primary key that was created by the CREATE TABLE statement or another ALTER TABLE statement.
- **foreign-key** Use this subclause to add a foreign key as a constraint. If you use a subclause other than ADD FOREIGN KEY with the ALTER TABLE statement on a table with dependent materialized views, the ALTER TABLE statement fails. For all other clauses, you must disable the dependent materialized views and then re-enable them when your changes are complete.

You can specify a MATCH subclause to control what is considered a match when using a multi-column foreign key. It also allows you to specify uniqueness for the key, thereby eliminating the need to declare uniqueness separately.

ADD PCTFREE clause Specify the percentage of free space you want to reserve in each table page. The free space is used if rows increase in size when the data is updated. If there is no free space in a table page, every increase in the size of a row on that page requires the row to be split across multiple table pages, causing row fragmentation and possible performance degradation. A free space percentage of 0 specifies that no free space is to be left on each page. Each page is to be fully packed. A high free space percentage causes each row to be inserted into a page by itself. If PCTFREE is not set, or is dropped, the default PCTFREE value is applied according to the database page size (200 bytes for a 4 KB or larger page size). The value for PCTFREE is stored in the ISYSTAB system table. When PCTFREE is set, all subsequent inserts into table pages use the new value, but rows that were already inserted are not affected. The value persists until it is changed. The PCTFREE specification can be used for base, global temporary, or local temporary tables.

Altering clauses There are several clauses you can use to alter the definition for a column

or table:

ALTER *column-name* *column-alteration* clause Use this clause to change attributes for the specified column. If a column is contained in a unique constraint, a foreign key, or a primary key, you can change only the default for the column. However, for any other change, you must delete the key or constraint before the column can be modified.

***column-data-type* clause** Use this clause to alter the length or data type of the column. Specify the new data type, or use the %TYPE attribute to set the data type to the data type of a column in another table or view. If necessary, then the data in the modified column is converted to the new data type. If a conversion error occurs, then the operation fails and the table is left unchanged. You cannot reduce the size of a column. For example, you cannot change a column from a VARCHAR(100) to a VARCHAR(50).

[NOT] NULL clause Use this clause to change whether NULLs are allowed in the column. If NOT NULL is specified, and the column value is NULL in any of the existing rows, then the operation fails and the table is left unchanged.

CHECK NULL Use this clause to delete all check constraints for the column.

DEFAULT clause Use this clause to change the default value for the column.

DEFAULT NULL clause Use this clause to remove the default value for the column.

[CONSTRAINT *constraint-name*] CHECK { NULL | (*condition*) } clause Use this clause to add a CHECK constraint on the column.

If you need to create a CHECK constraint that involves a relationship between two or more columns in the table (for example, column A must be less than column B), define a table constraint instead.

[NOT] COMPRESSED clause Use this clause to change whether the column is compressed.

INLINE and PREFIX clauses The INLINE clause specifies the maximum BLOB size, in bytes, to store within the row. BLOBs smaller than or equal to the value specified by the INLINE clause are stored within the row. BLOBs that exceed the value specified by the INLINE clause are stored outside the row in table extension pages. Also, a copy of some bytes from the beginning of the BLOB may be kept in the row when a BLOB is larger than the INLINE value. Use the PREFIX clause to specify how many bytes are kept in the row. The PREFIX clause can improve the performance of requests that need the prefix bytes of a BLOB to determine if a row is accepted or rejected.

The prefix data for a compressed column is stored uncompressed, so if all the data required to satisfy a request is stored in the prefix, no decompression is necessary.

If neither INLINE nor PREFIX is specified, or if USE DEFAULT is specified, default values are applied as follows:

- For character data type columns, such as CHAR, NCHAR, and LONG VARCHAR, the default value of INLINE is 256, and the default value of PREFIX is 8.
- For binary data type columns, such as BINARY, LONG BINARY, VARBINARY, BIT, VARBIT, LONG VARBIT, BIT VARYING, and UUID, the default value of INLINE is 256, and the default value of PREFIX is 0.

Note It is strongly recommended that you use the default values unless there are specific circumstances that require a different setting. The default values have been chosen to balance performance and disk space requirements. For example,

if you set **INLINE** to a large value, and all the BLOBs are stored inline, row processing performance may degrade. If you set **PREFIX** too high, you increase the amount of disk space required to store BLOBs since the prefix data is a duplicate of a portion of the BLOB.

If only one of the values is specified, the other value is automatically set to the largest amount that does not conflict with the specified value. Neither the **INLINE** nor **PREFIX** value can exceed the database page size. Also, there is a small amount of overhead reserved in a table page that cannot be used to store row data. Therefore, specifying an **INLINE** value approximate to the database page size can result in a slightly smaller number of bytes being stored inline.

INDEX and NO INDEX clauses When storing BLOBs (character or binary types only), specify **INDEX** to create BLOB indexes on inserted values that exceed the internal BLOB size threshold (approximately eight database pages). This is the default behavior.

BLOB indexes can improve performance when random access searches within the BLOBs are required. However, for some types of BLOB values, such as images and multimedia files that will never require random-access, performance can improve if BLOB indexing is turned off. To turn off BLOB indexing for a column, specify **NO INDEX**.

Note A BLOB index is not the same as a table index. A table index is created to index values in one or more columns.

SET COMPUTE clause When a column is created using a **COMPUTE** clause, its value in any row is the value of the supplied expression. Columns created with this constraint are read-only columns for applications: the value is changed by the database server whenever the row is modified. The **COMPUTE** expression should not return a non-deterministic value. For example, it should not include a special value such as **CURRENT TIMESTAMP**, or a non-deterministic function. If a **COMPUTE** expression returns a non-deterministic value, then it cannot be used to match an expression in a query.

The **COMPUTE** clause is ignored for remote tables.

Any **UPDATE** statement that attempts to change the value of a computed column fires any triggers associated with the column.

ALTER CONSTRAINT *constraint-name* CHECK clause Use this clause to alter a named check constraint for the table.

To alter the constraint to specify a relationship between two or more columns in the table (for example, column A must be less than column B), define a table constraint instead.

Dropping clauses

DROP DEFAULT Drops the default value set for the table or specified column. Existing values do not change.

DROP COMPUTE Removes the **COMPUTE** attribute for the specified column. This statement does not change any existing values in the table.

DROP CHECK Drops all **CHECK** constraints for the table or specified column. **DELETE CHECK** is also accepted.

DROP CONSTRAINT *constraint-name* Drops the named constraint for the table or specified column. **DELETE CONSTRAINT** is also accepted.

DROP *column-name* Drops the specified column from the table. **DELETE *column-name***

is also accepted. If the column is contained in any index, unique 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.

DROP UNIQUE (*column-name* ...) Drop the unique constraints on the specified column(s). Any foreign keys referencing this unique constraint are also deleted. DELETE UNIQUE (*column-name* ...) is also accepted.

DROP FOREIGN KEY *fkey-name* Drop the specified foreign key. DELETE FOREIGN KEY *fkey-name* is also accepted.

DROP PRIMARY KEY Drop the primary key. All foreign keys referencing the primary key for this table are also deleted. DELETE PRIMARY KEY is also accepted.

Renaming clauses

RENAME *new-table-name* Change the name of the table to *new-table-name*. Any applications using the old table name must be modified, as necessary.

RENAME *column-name* TO *new-column-name* Change the name of the column to the *new-column-name*. Any applications using the old column name must be modified, as necessary.

RENAME CONSTRAINT *constraint-name* TO *new-constraint-name* Change the name of the constraint to the *new-constraint-name*.

ALTER TABLE...RENAME CONSTRAINT *constraint-name* TO *new-constraint-name*, when used for an RI constraint, only renames the constraint, not the underlying index or, if applicable, the foreign key role name. To rename the underlying index or the role name, use the ALTER INDEX statement.

table-alteration clauses Use this clause to alter the following table attributes.

PCTFREE DEFAULT Use this clause to change the percent free setting for the table to the default (200 bytes for a 4 KB, and up, page size).

[NOT] ENCRYPTED Use this clause to change whether the table is encrypted. To encrypt a table, table encryption must already be enabled on the database. The table is encrypted using the encryption key and algorithm specified at database creation time.

After encrypting a table, any data for that table that was in temporary files or the transaction log before encryption still exists in unencrypted form. To address this, restart the database to remove the temporary files. Run the Backup utility (dbbackup) with the -o option, or use the BACKUP DATABASE statement, to back up the transaction log and start a new one.

When table encryption is enabled, table pages for the encrypted table, associated index pages, temporary file pages, and transaction log pages containing transactions on encrypted tables are encrypted.

DISABLE VIEW DEPENDENCIES clause Use this clause to disable dependent regular views. Dependent materialized views are not disabled; you must disable each dependent materialized view by executing an ALTER MATERIALIZED VIEW...DISABLE statement.

ALTER OWNER clause To alter the owner of a table:

- You must have the ALTER ANY OBJECT OWNER privilege.
- You must have one of the following privileges: ALTER privilege on the table, ALTER ANY TABLE privilege, or ALTER ANY OBJECT privilege.
- The new owner cannot already own a table with the same name.
- Enabled materialized views cannot reference the table.

PRESERVE or DROP PRIVILEGES If you do not want the new owner to have the same privileges as the old owner, you can specify DROP PRIVILEGES to drop all explicitly granted privileges that allow a user access to the table. Implicitly granted privileges given to the owner of the table are given to the new owner and dropped from the old owner.

PRESERVE or DROP FOREIGN KEYS To prevent the new owner from accessing data in referenced tables, you can specify DROP FOREIGN KEYS to drop all foreign keys within the table, as well as all foreign keys referring to the table.

Remarks

The ALTER TABLE statement changes table attributes (column definitions, constraints, and so on) in an existing table.

The database server keeps track of object dependencies in the database. Alterations to the schema of a table may impact dependent views. Also, if there are materialized views that are dependent on the table you are attempting to alter, you must first disable them using the ALTER MATERIALIZED VIEW...DISABLE statement.

You cannot use ALTER TABLE on a local temporary table.

ALTER TABLE is prevented whenever the statement affects a table that is currently being used by another connection. ALTER TABLE can be time-consuming, and the database server does not process other requests referencing the table while the statement is being processed.

If you alter a column that a text index defined as IMMEDIATE REFRESH is built on, the text index is immediately rebuilt. If the text index is defined as AUTO REFRESH or MANUAL REFRESH, the text index is rebuilt the next time it is refreshed.

When you execute an ALTER TABLE statement, the database server attempts to restore column privileges on dependent views that are automatically recompiled. Privileges on columns that no longer exist in the recompiled views are lost.

ALTER TABLE requires exclusive access to the table.

Global temporary tables cannot be altered unless all users that have referenced the temporary table have disconnected.

This statement cannot be used within a snapshot transaction.

Privileges

You must be the owner of the table, or have one of the following privileges:

- ALTER privilege on the table
- ALTER ANY TABLE system privilege
- ALTER ANY OBJECT system privilege
- MANAGE ANY DBSPACE system privilege

To alter the table owner, you must also have the ALTER ANY OBJECT OWNER system privilege.

To create, alter, or drop indexes on a table, you must have the CREATE ANY INDEX, ALTER ANY INDEX, or DROP ANY INDEX system privilege, respectively.

Side effects

Automatic commit.

A checkpoint is carried out at the beginning of the ALTER TABLE operation, and further checkpoints are suspended until the ALTER operation completes.

Once you alter a column or table, any stored procedures, views, or other items that refer to the altered column may no longer work.

If you change the declared length or type of a column, or drop a column, the statistics for that column are dropped.

Standards

- **ANSI/ISO SQL Standard** Core Feature. In the ANSI/ISO SQL Standard, ADD COLUMN and DROP COLUMN are supported as Core Features, as are ADD CONSTRAINT and DROP CONSTRAINT. ALTER [COLUMN] is SQL Feature F381, as is the ability to add, modify, or drop a DEFAULT value for a column. In the ANSI/ISO SQL Standard, altering the data type of a column is performed by specifying the SET DATA TYPE clause, which is SQL Language Feature F382. Conversely, the software supports modifying a column's data type through the ALTER clause directly.

Other clauses supported by the software, including ALTER CONSTRAINT, RENAME, PCTFREE, ENCRYPTED, and DISABLE MATERIALIZED VIEW, are not in the standard. Support for extensions to column definitions, and column and table constraint definitions, are either not in the standard, or are optional features of the standard.

- **Transact-SQL** ALTER TABLE is supported by Adaptive Server Enterprise. Adaptive Server Enterprise supports the ADD COLUMN and DROP COLUMN clauses, in addition to ADD CONSTRAINT and DROP CONSTRAINT. Adaptive Server Enterprise uses MODIFY rather than the keyword ALTER for the ALTER clause. Adaptive Server Enterprise uses the REPLACE clause for altering a column's DEFAULT value. In Adaptive Server Enterprise, ALTER TABLE is also used to enable/disable triggers for a specific table, a feature that is not supported in the software.

Example

The following example adds a new timestamp column, TimeStamp, to the Customers table. To run this next statement, you must have the ALTER privilege on the Customers table.

```
ALTER TABLE GROUP0.Customers  
ADD TimeStamp AS TIMESTAMP DEFAULT TIME;--
```

The following example drops the new timestamp column, TimeStamp that you added in the previous example.

```
ALTER TABLE GROUP0.Customers  
DROP TimeStamp;
```

The Street column in the Customers table can currently hold up to 35 characters. To allow it to hold up to 50 characters, execute the following:

```
ALTER TABLE GROUP0.Customers  
ALTER Street CHAR(50);
```

The following example adds a column to the Customers table, assigning each customer a sales contact. To run this next statement, you must also have the CREATE ANY INDEX system privilege because it creates a foreign key.

```
ALTER TABLE GROUP0.Customers
ADD SalesContact INTEGER
REFERENCES GROUP0.Employees ( EmployeeID )
ON UPDATE CASCADE
ON DELETE SET NULL;
```

This foreign key is constructed with cascading updates and is set to 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.

The following example creates a foreign key, FK_SalesRepresentative_EmployeeID2, on the SalesOrders.SalesRepresentative column, linking it to Employees.EmployeeID. You must have the ALTER privilege on the SalesOrder table to execute the following statement:

```
ALTER TABLE GROUP0.SalesOrders
ADD CONSTRAINT FK_SalesRepresentative_EmployeeID2
FOREIGN KEY ( SalesRepresentative )
REFERENCES GROUP0.Employees (EmployeeID);
```

The following example adds a column where the default is AUTOINCREMENT. In this example, all existing customer rows are modified to have a nullable AUTOINCREMENT column with the column value assigned, but the database server does not guarantee which row is assigned which value:

```
ALTER TABLE GROUP0.Customers
ADD Surrogate_key INTEGER DEFAULT AUTOINCREMENT;
```

The following example changes the owner of the fictitious table mytable to Bob:

```
ALTER TABLE mytable ALTER OWNER TO bob;
```

Related Information

[Strings](#)

A string is a sequence of characters up to 2 GB in size.

[Events](#)

You can automate routine tasks by adding an event to a database, and providing a schedule for the event.

[Reloading tables with AUTOINCREMENT columns](#)

You can retain the next available value for AUTOINCREMENT columns in the rebuilt database by specifying the dbunload -I option. This option adds calls to the sa_reset_identity system procedure to the generated *reload.sql* script for each table that contains an AUTOINCREMENT value, preserving the current value of SYSTABCOL.max_identity.

[Table alteration](#)

Alter the structure or column definitions of a table by adding columns, changing various column attributes, or deleting columns.

[The special IDENTITY column](#)

The value of the IDENTITY column uniquely identifies each row in a table.

[Special values](#)

Special values can be used in expressions, and as column defaults when creating tables.

[Table and column constraints](#)

The CREATE TABLE statement and ALTER TABLE statement allow you to specify table attributes that allow control over data accuracy and integrity.

[Use of a sequence to generate unique values](#)

You can use a **sequence** to generate values that are unique across multiple tables or that are different from a set of natural numbers.

[Clustered indexes](#)

You can improve the performance of a large index scan by declaring that the index is **clustered**.

[View dependencies](#)

A view definition refers to other objects such as columns, tables, and other views, and these references make the view **dependent** on the objects to which it refers.

[%TYPE and %ROWTYPE attributes](#)

In addition to explicitly setting the data type for an object, you can also set the data type by specifying the %TYPE and %ROWTYPE attributes.

[Encrypting a table](#)

Create an encrypted table, or encrypt an existing table.

[sa_reset_identity system procedure](#)

Allows the next identity value to be set for a table. Use this procedure to change the AUTOINCREMENT value for the next row that will be inserted.

[ALTER MATERIALIZED VIEW statement](#)

Alters a materialized view.

[BACKUP DATABASE statement](#)

Backs up a database and transaction log.

[ALTER INDEX statement](#)

Renames an index, primary key, or foreign key, or changes the clustered nature of an index.

[CREATE TABLE statement](#)

Creates a new table in the database and, optionally, creates a table on a remote server.

[DROP TABLE statement](#)

Removes a table from the database.

[Backup utility \(dbbackup\)](#)

Creates a client-side or a server-side backup of database files and transaction logs for running databases.

[SQL data types](#)

There are many SQL data types supported by the software.

[allow_nulls_by_default option](#)

Controls whether new columns that are created without specifying either NULL or NOT NULL are allowed to contain NULL values.

[SQL variables](#)

The supported variables can be grouped by scope: connection, database, and global.

[@@identity global variable](#)

The @@identity variable holds the most recent value inserted by the current connection into an IDENTITY column, a DEFAULT AUTOINCREMENT column, or a DEFAULT GLOBAL AUTOINCREMENT column, or zero if the most recent insert was into a table that had no such column.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

ALTER TEXT CONFIGURATION statement

Alters a text configuration object.

Syntax

```
ALTER TEXT CONFIGURATION [ owner.]config-name
STOPLIST stoplist-string
| DROP STOPLIST
| { MINIMUM | MAXIMUM } TERM LENGTH integer }
| TERM BREAKER { GENERIC [ EXTERNAL NAME external-call ] | NGRAM }
| PREFILTER EXTERNAL NAME external-call
| DROP PREFILTER
| SAVE OPTION VALUES [ FROM CONNECTION ]

external-call :
[ system-configuration:]function-name@library-file-prefix[.{ so | dll } ]

system-configuration :
{ generic-operating-system | specific-operating-system } [ (processor-architecture) ]

generic-operating-system :
{ Unix | Windows }

specific-operating-system :
{ AIX | HPUX | Linux | OSX | Solaris | WindowsNT }

processor-architecture :
{ 32 | 64 | ARM | IA64 | PPC | SPARC | X86 | X86_64 }
```

Parameters

STOPLIST clause Use this clause to create or replace the list of terms to ignore when building a text index. Using this text configuration object, terms specified in this list are also ignored in a query. Separate stoplist terms with spaces. For example, `STOPLIST 'because about therefore only'`. Stoplist terms cannot contain whitespace.

Samples of stoplists for different languages are located in %SQLANYSAMPI7%\SQLAnywhere\SQL.

Stoplist terms should not contain non-alphanumeric characters. The stoplist length must be less than 8000 bytes.

Carefully consider whether you want to put terms in your stoplist.

DROP STOPLIST clause Use this clause to drop the stoplist for a text configuration object.

MINIMUM TERM LENGTH clause MINIMUM TERM LENGTH clause is ignored when using NGRAM text indexes.

The minimum length, in characters, of a term to include in the text index. Terms that are shorter than this setting are ignored when building or refreshing the text index. The value of this option must be greater than 0. If you set this option to be higher than MAXIMUM TERM LENGTH, the

value of MAXIMUM TERM LENGTH is automatically adjusted to be the same as the new MINIMUM TERM LENGTH value.

MAXIMUM TERM LENGTH clause With NGRAM text indexes, use the MAXIMUM TERM LENGTH clause to set the size of the n-grams into which strings are broken. Terms shorter than MAXIMUM TERM LENGTH are not indexed.

With GENERIC text indexes, use the MAXIMUM TERM LENGTH clause to set the maximum length, in characters, of a term to include in the text index. Terms that are longer than this setting are ignored when building or refreshing the text index. The value of MAXIMUM TERM LENGTH must be less than or equal to 60. If you set this option to be lower than MINIMUM TERM LENGTH, the value of MINIMUM TERM LENGTH is automatically adjusted to be the same as the new MAXIMUM TERM LENGTH value.

TERM BREAKER clause The name of the algorithm to use for separating column values into terms. The choices are GENERIC (the default) or NGRAM.

- o **GENERIC** For GENERIC, you can use the built-in GENERIC term breaker algorithm by specifying TERM BREAKER GENERIC, or you can specify an external algorithm using the TERM BREAKER GENERIC EXTERNAL NAME clause.

The built-in GENERIC algorithm treats any string of one or more alphanumerics, separated by non-alphanumerics, as a term.

Specify the TERM BREAKER GENERIC EXTERNAL NAME clause to specify an entry point to a term breaker function in an external library. This is useful if you have custom requirements for how you want terms broken up before they are indexed or queried (for example, if you want an apostrophe to be considered as part of a term and not as a term breaker).

external-call can specify more than one function and/or library, and can include the file extension of the library, which is typically *.dll* on Windows, and *.so* on Unix. In the absence of the file extension, the database server defaults to the platform-specific file extension for libraries. For example, `EXTERNAL NAME`

`'TermBreakFunc1@myTLib;Unix:TermBreakFunc2@myTLib'` calls the TermBreakFunc1 function from *myTLib.dll* on Windows, and the TermBreakFunc2 function from *myTLib.so* on Unix.

- o **NGRAM** The built-in NGRAM algorithm breaks strings into n-grams. An n-gram is an *n*-character substring of a larger string. The NGRAM term breaker is required for fuzzy (approximate) matching, or for documents that do not use whitespace or non-alphanumeric characters to separate terms, if no external term breaker is specified.

PREFILTER EXTERNAL NAME clause Specify the PREFILTER EXTERNAL NAME clause to specify an entry point to a prefilter function in an external library. This is useful if text data needs to be extracted from binary data (for example, PDF). It is also useful if the text you want to index contains formatting information and/or images that you want to strip out before indexing the data (for example, HTML).

Because *external-call* can specify multiple sets of operating systems, processors, libraries, and functions, more-precisely specified configurations take precedence over less-precisely defined configurations. For example, `Solaris(X86_64):myfunc64@mylib.so` takes precedence over `Solaris:myfunc64@mylib.so`, and so on.

For syntaxes that support *system-configuration*, if you do not specify *system-configuration*, then it is assumed that the procedure runs on all system configurations. Unix represents the following Unix-based operating systems: AIX, HP-UX, Linux, OSX, and Solaris. The generic term Windows represents all versions of the Windows operating system.

If you specify Unix for one of the calls, then it is assumed that the other call is for Windows.

The *specific-operating-system* and *processor-architecture* values are those operating systems and processors supported by SQL Anywhere server.

The library name (*library-file-prefix*) is followed by the file extension, which is typically *.dll* on Windows and *.so* on Unix. In the absence of the file extension, the database server defaults to the platform-specific file extension for libraries. For example, `PREFILTER EXTERNAL NAME 'PrefilterFunct1@myPreFilterlib;Unix:PrefilterFunct2@myPreFilterlib'` calls the PrefilterFunct1 function from *myPreFilterlib.dll* on Windows, and the PrefilterFunct2 function from *myPreFilterlib.so* on Unix.

DROP PREFILTER clause Use the DROP PREFILTER clause to drop use of the specified prefiltering library for the text configuration object. Prefiltering is no longer performed when the database server builds indexes that use this text configuration object.

SAVE OPTION VALUES clause When a text configuration object is created, the current `date_format`, `time_format`, `timestamp_format`, and `timestamp_with_time_zone_format` database options reflect how DATE, TIME, and TIMESTAMP columns are saved with the text configuration object. Use the SAVE OPTION VALUES clause to update the option values saved for the text configuration object to reflect the options currently in effect for the connection.

Remarks

Before changing the term length settings, read about the impact of various settings on what gets indexed and how query terms are interpreted.

Text indexes are dependent on a text configuration object. Before using this statement you must truncate dependent AUTO or MANUAL REFRESH text indexes, and drop any IMMEDIATE REFRESH text indexes.

To view the settings for text configuration objects, query the SYSTEXTCONFIG system view.

Privileges

You must be the owner of the text configuration object, or have one of the following privileges:

- ALTER privilege on the text configuration object
- ALTER ANY TEXT CONFIGURATION system privilege
- ALTER ANY OBJECT system privilege

If you are altering an external term breaker, you must also have the CREATE EXTERNAL REFERENCE system privilege.

When specifying or dropping a prefilter, or specifying an external term breaker, you must *also* have the ALTER ANY TEXT CONFIGURATION or ALTER ANY OBJECT system privilege.

Side effects

Automatic commit

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statements create a text configuration object, `myTextConfig`, and then change the maximum term length to 16. To run this statement, you'll need the CREATE TEXT CONFIGURATION system privilege:

```
CREATE TEXT CONFIGURATION myTextConfig FROM default_char;  
ALTER TEXT CONFIGURATION maxTerm16  
    MAXIMUM TERM LENGTH 16;
```

The following statement adds a stoplist to the myTextConfig configuration object:

```
ALTER TEXT CONFIGURATION myTextConfig  
    STOPLIST 'because about therefore only';
```

The following statement configures an external term breaker for the myTextConfig text configuration object. Both the Windows and Unix interfaces are specified.

```
ALTER TEXT CONFIGURATION myTextConfig  
    TERM BREAKER GENERIC  
    EXTERNAL NAME  
'my_termbreaker@termbreaker.dll;Unix:my_termbreaker@libtermbreaker_r.so'
```

The following example configures an external prefilter for the myTextConfig text configuration object. Both the Windows and Unix interfaces are specified.

```
ALTER TEXT CONFIGURATION myTextConfig  
    PREFILTER EXTERNAL NAME  
'html_xml_filter@html_xml_filter.dll;UNIX:html_xml_filter@libhtml_xml_filter_r.so';
```

The following example drops the external prefilter for the myTextConfig text configuration object.

```
ALTER TEXT CONFIGURATION myTextConfig DROP PREFILTER;
```

Related Information

[What to specify when creating or altering text configuration objects](#)

There are many settings to configure when creating or altering a text configuration object.

[Altering a text configuration object](#)

Alter text configuration object properties such as the term breaker type, the stoplist, and option settings.

[Viewing text index terms and settings \(SQL Central\)](#)

View text index terms and settings in SQL Central.

[Tutorial: Performing a full text search on a GENERIC text index](#)

Perform a full text search on a text index that uses a GENERIC term breaker.

[Tutorial: Performing a fuzzy full text search](#)

Perform a fuzzy full text search on a text index that uses an NGRAM term breaker.

[CREATE TEXT CONFIGURATION statement](#)

Creates a text configuration object for use with building and updating text indexes.

[DROP TEXT CONFIGURATION statement](#)

Drops a text configuration object.

[sa_char_terms system procedure](#)

Breaks a CHAR string into terms and returns each term as a row along with its position.

[sa_nchar_terms system procedure](#)

Breaks an NCHAR string into terms and returns each term as a row along with its position.

[sa_refresh_text_indexes system procedure](#)

Refreshes all text indexes defined as MANUAL REFRESH or AUTO REFRESH.

[sa_text_index_stats system procedure](#)

Returns statistical information about the text indexes in the database.

[SYSTEXTCONFIG system view](#)

Each row in the SYSTEXTCONFIG system view describes one text configuration object, for use with the full text search feature. The underlying system table for this view is ISYSTEXTCONFIG.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

ALTER TEXT INDEX statement

Alters the definition of a text index.

Syntax

```
ALTER TEXT INDEX [ owner.]text-index-name  
ON [ owner.]table-name  
alter-clause  
  
alter-clause :  
rename-object  
| refresh-alteration  
  
rename-object :  
RENAME { AS | TO } new-name  
  
refresh-alteration :  
{ MANUAL REFRESH  
| AUTO REFRESH [ EVERY integer { MINUTES | HOURS } ] }
```

Parameters

RENAME clause Use the RENAME clause to rename the text index.
REFRESH clause Specify the REFRESH clause to set the refresh type for the text index.

Remarks

Once a text index is created, you cannot change it to, or from, IMMEDIATE REFRESH. If either of these changes is required, you must drop and recreate the text index.

This statement cannot be executed when there are cursors opened with the WITH HOLD clause that use either statement or transaction snapshots.

You can only alter a text index built on a materialized view by renaming it. You cannot change the refresh type for a text index built on a materialized view.

Privileges

To alter a text index on a table, you must be the owner of the table, or have one of the following privileges:

- REFERENCES privilege on the table
- ALTER ANY INDEX system privilege
- ALTER ANY OBJECT system privilege

To alter a text index on a materialized view, you must be the owner of the materialized, or have one of the following privileges:

- ALTER ANY INDEX system privilege
- ALTER ANY OBJECT system privilege

Side effects

Automatic commit

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The first statement creates a text index, txt_index_manual, defining it as MANUAL REFRESH. The second statement alters the text index to refresh automatically every day. The third statement renames the text index to txt_index_daily.

```
CREATE TEXT INDEX txt_index_manual ON GROUP0.MarketingInformation ( Description )  
    MANUAL REFRESH;  
ALTER TEXT INDEX txt_index_manual ON GROUP0.MarketingInformation  
    AUTO REFRESH EVERY 24 HOURS;  
ALTER TEXT INDEX txt_index_manual ON GROUP0.MarketingInformation  
    RENAME AS txt_index_daily;
```

Related Information

[Snapshot isolation](#)

Snapshot isolation is designed to improve concurrency and consistency by maintaining different versions of data.

[Full text search](#)

You can perform full text searching on tables.

[Viewing text index terms and settings \(SQL Central\)](#)

View text index terms and settings in SQL Central.

[Tutorial: Performing a full text search on a GENERIC text index](#)

Perform a full text search on a text index that uses a GENERIC term breaker.

[Tutorial: Performing a fuzzy full text search](#)

Perform a fuzzy full text search on a text index that uses an NGRAM term breaker.

[CREATE TEXT INDEX statement](#)

Creates a text index.

[ALTER TEXT INDEX statement](#)

Alters the definition of a text index.

[DROP TEXT INDEX statement](#)

Removes a text index from the database.

[REFRESH TEXT INDEX statement](#)

Refreshes a text index.

[TRUNCATE TEXT INDEX statement](#)

Deletes the data in a MANUAL or an AUTO REFRESH text index.

[COMMENT statement](#)

Stores a comment for a database object in the system tables.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

ALTER TIME ZONE statement

Modifies a time zone object.

Syntax

```
ALTER TIME ZONE name time-zone-option [ ... ]
```

time-zone-option :

```
{ OFFSET offset  
| DST OFFSET offset  
| NO DST  
| [ DST ] STARTING month-day-rule AT { minutes | hours:minutes }  
| [ DST ] ENDING month-day-rule AT { minutes | hours:minutes } }
```

Remarks

All clauses are optional; however, if none are present, then the statement fails.

If you are adding daylight savings time to a time zone that previously had no daylight savings time, then use the STARTING and ENDING clauses. If you are removing daylight savings time from a time zone, then specify the NO DST clause and omit the STARTING and ENDING clauses.

Privileges

You must have the MANAGE TIME ZONE system privilege or the ALTER ANY OBJECT system privilege.

Side effects

Automatic commit.

Executing this statement populates the ISYSTIMEZONE system table.

Example

To change the time zone named EasternTime, which uses daylight savings time, to use Australian Eastern Time without daylight savings time, execute the following statement:

```
ALTER TIME ZONE EasternTime OFFSET '10:00'  
NO DST;
```

Related Information

[COMMENT statement](#)

Stores a comment for a database object in the system tables.

[CREATE TIME ZONE statement](#)

Creates a simulated time zone that can be used by any database.

[DROP TIME ZONE statement](#)

Drops a time zone from the database.

[time_zone option](#)

Specifies which time zone the database uses for time zone calculations.

[SYSTIMEZONE system view](#)

Each row in the SYSTIMEZONE system view describes one time zone. The underlying system table for this view is ISYSTIMEZONE.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

ALTER TRACE EVENT SESSION statement

Adds or removes trace events from a session, adds or removes targets from a session, or starts or stops a trace session.

Syntax

```
ALTER TRACE EVENT SESSION session-name
[ ON SERVER ]
{ ADD TRACE EVENT trace-event-name [...]  
  | DROP TRACE EVENT trace-event-name [...]  
  | ADD TARGET FILE [ ( SET target-parameter-name=target-parameter-value [, ...] ) ]  
  | DROP TARGET target-name [, ...] ]  
  | STATE = { START | STOP }  
}
```

target-parameter-name :

```
{ filename_prefix  
  | max_size  
  | num_files  
  | flush_on_write  
  | compressed }
```

Parameters

ON SERVER clause Alters a trace event session that is recording trace events from all databases on the database server. If this clause is not specified, then only the trace event session on the current database is altered.

ADD TRACE EVENT The name of the trace event being added to the session.

DROP TRACE EVENT The name of the trace event being removed from the session.

ADD TARGET The name of the target being added to the session. Information about the trace events that are part of the session is logged to this target (file).

DROP TARGET The name of the target being removed from the session.

Remarks

Adding or dropping trace events or targets from a running trace session causes the session to pause briefly to make the changes and then resume after the changes are made. You do not need to stop a tracing session to add or drop trace events or targets. Adding or removing a trace event from a session that has already started has the side effect that some trace events are missed while a session is temporarily paused.

System privileges

You must have the **MANAGE ANY TRACE SESSION** system privilege.

Side effects

None

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example creates a trace event called my_event, creates a trace event session called my_session, and starts the session:

```
CREATE TEMPORARY TRACE EVENT my_event( id INTEGER, information LONG VARCHAR );
CREATE TEMPORARY TRACE EVENT SESSION my_session
    ADD TRACE EVENT my_event, -- user event
    ADD TRACE EVENT SYS_ConsoleLog_Information -- system event
    ADD TARGET FILE ( SET filename_prefix='my_trace_file' ); -- add a target
ALTER TRACE EVENT SESSION my_session
    STATE = START;
```

Related Information

[Event tracing](#)

Event tracing records information about system-defined and user-defined trace events to an event trace data (ETD) file.

[System events](#)

Each system event provides a hook on which you can program a set of actions.

[CREATE TEMPORARY TRACE EVENT statement](#)

Creates a user trace event that persists until the database is stopped.

[CREATE TEMPORARY TRACE EVENT SESSION statement](#)

Creates a user trace event session.

[DROP TRACE EVENT statement](#)

Drops a user-defined trace event.

[DROP TRACE EVENT SESSION statement](#)

Drops a trace event session.

[NOTIFY TRACE EVENT statement](#)

Logs a user-defined trace event to a trace session.

[sp_trace_events system procedure](#)

Returns information about the trace events in the database.

[sp_trace_event_fields system procedure](#)

Returns information about the fields of the specified trace event.

[sp_trace_event_sessions system procedure](#)

Returns a list of the trace event sessions that are defined for the database.

[sp_trace_event_session_events system procedure](#)

Lists the trace events that are part of a specific trace event session.

[sp_trace_event_session_targets system procedure](#)

Lists the targets of a trace session.

[sp_trace_event_session_target_options system procedure](#)

Lists the target options for a trace event session.

[Event Trace Data \(ETD\) File Management utility \(dbmanagetd\)](#)

Generates a diagnostic log that allows the user to examine information for user-defined and system events.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

ALTER TRIGGER statement

Replaces a trigger definition with a modified version. You must include the entire new trigger definition in the ALTER TRIGGER statement.

Syntax

- **Change the definition of a trigger**

ALTER TRIGGER *trigger-name trigger-definition*

trigger-definition : [CREATE TRIGGER syntax](#)

- **Obfuscate a trigger definition**

ALTER TRIGGER *trigger-name* **ON** [*owner.*] *table-name* **SET HIDDEN**

Remarks

- **Change the definition of a trigger** The ALTER TRIGGER statement is identical in syntax to the CREATE TRIGGER statement except for the first word.
Either the Transact-SQL or Watcom SQL form of the CREATE TRIGGER syntax can be used.
- **Obfuscate a trigger definition** You can use SET HIDDEN to obfuscate the definition of the associated trigger and cause it to become unreadable. The trigger can be unloaded and reloaded into other databases. If SET HIDDEN is used, debugging using the debugger does not show the trigger definition, nor is it available through procedure profiling.

Note The SET HIDDEN operation is irreversible.

Privileges

You must be the owner of the underlying table, or have the one of the following privileges:

- ALTER privilege on the underlying table with the CREATE ANY OBJECT system privilege
- ALTER ANY TRIGGER system privilege
- ALTER ANY OBJECT system privilege

To alter a trigger on a view owned by someone else, you must have either the ALTER ANY TRIGGER and ALTER ANY VIEW system privileges, or you must have the ALTER ANY OBJECT system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Related Information

[Hiding the contents of a procedure, function, trigger, event, or view](#)

Use the SET HIDDEN clause to obscure the contents of a procedure, function, trigger, event, or view.

[CREATE TRIGGER statement](#)

Creates a trigger on a table.

[CREATE TRIGGER statement \[T-SQL\]](#)

Creates a new trigger in the database in a manner compatible with Adaptive Server Enterprise.

[DROP TRIGGER statement](#)

Removes a trigger from the database.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

ALTER USER statement

Alters user settings.

Syntax

- **Change the definition of a database user**

```
ALTER USER user-name  
[ IDENTIFIED BY password ]  
[ LOGIN POLICY policy-name ]  
[ FORCE PASSWORD CHANGE { ON | OFF } ]
```

- **Unlock a database user**

```
ALTER USER user-name  
[ RESET LOGIN POLICY ]
```

- **Refresh the Distinguished Name (DN) for an LDAP user**

```
ALTER USER user-name  
REFRESH DN
```

- **Change a password part**

```
ALTER USER user-name  
[ IDENTIFIED { FIRST | LAST } BY password-part ]
```

Parameters

user-name The name of the user.

IDENTIFIED clause The password of the user. A user without a password cannot connect to the database. This is useful if you are creating a group and do not want anyone to connect to the database using the group user ID.

- **IDENTIFIED BY clause** Use this clause to reset the password for a user. To reset a user to not have a password, set *password* to NULL.
- **IDENTIFIED { FIRST | LAST } BY clause** Use this clause to reset part of the password for a user who has a dual control password. A user with dual control password has the CHANGE_PASSWORD_DUAL_CONTROL login policy option enabled for their login policy.

Two administrators are required to reset a dual control password. One administrator executes the IDENTIFIED FIRST BY clause to set the first part of the password and another administrator executes the IDENTIFIED LAST BY clause to set the last part of the password. The user combines the two password parts and uses this combined password to connect to the database.

policy-name The name of the login policy to assign the user. No change is made if the LOGIN POLICY clause is not specified.

FORCE PASSWORD CHANGE clause Controls whether the user must specify a new password when they log in. This setting overrides the password_expiry_on_next_login option setting in the user's policy.

RESET LOGIN POLICY clause Resets the number of failed login attempts, as well as the

user's last login time and last failed login time. If a user account was locked for exceeding login policy limits, it is unlocked.

REFRESH DN clause REFRESH DN clears the Distinguished Name (DN) and timestamp of the user so that at the time of the next LDAP authentication, the search for the DN is done. If the authentication succeeds during the next LDAP authentication of this user then both the DN and the timestamp are updated with the new DN and current time.

Remarks

The following list describes the requirements for user IDs and passwords. The requirements and restrictions for a password part are the same as those described for a password except that the maximum length of each part is 127 bytes.

The `verify_password_function` login policy option can be used to specify a function to implement password rules (for example, passwords must include at least one digit). If a password verification function is used, you cannot specify more than one user ID and password in the GRANT CONNECT statement.

If you set the `password_expiry_on_next_login` value to ON, the user's password expires immediately when they next login even if they are assigned to the same policy. You can use the ALTER USER and LOGIN POLICY clauses to force a user to change their password when they next login.

The ALTER USER...REFRESH DN syntax clears the Distinguished Name (DN) and timestamp of a user so that during the next LDAP authentication, a search for the DN is performed, instead of using the cached DN, which can become out of date. If the authentication succeeds, then both the DN and the timestamp are updated with the new DN and current time.

If you use this statement in a procedure, do not specify the password (IDENTIFIED BY clause) as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

Privileges

Any user can change their own password.

To change passwords for other users, you must have the CHANGE PASSWORD system privilege.

For all other changes to other users, including forcing users to change their password, you must have the MANAGE ANY USER system privilege.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statement alters a user named SQLTester, setting their password to welcome123, setting their login policy to Test1, and allowing them to bypass a forced password change.

```
ALTER USER SQLTester IDENTIFIED BY welcome123
LOGIN POLICY Test1
FORCE PASSWORD CHANGE off;
```

The following example refreshes the LDAP Distinguished Name for user myusername.

```
ALTER USER myusername REFRESH DN;
```

Related Information

[Dual control passwords](#)

The dual control password feature requires two administrators to change a password.

[Login policies](#)

A **login policy** consists of a set of rules that are applied when you create a database connection for a user.

[Assigning a login policy to an existing user](#)

Change the login policy that is assigned to a user by assigning the user a new login policy.

[ALTER LOGIN POLICY statement](#)

Alters an existing login policy.

[COMMENT statement](#)

Stores a comment for a database object in the system tables.

[CREATE LOGIN POLICY statement](#)

Creates a login policy.

[CREATE USER statement](#)

Creates a database user or group.

[DROP LOGIN POLICY statement](#)

Drops a login policy.

[DROP USER statement](#)

Drops a user.

[verify_password_function option](#)

Implements password rules.

[GRANT CONNECT statement](#)

Creates a new user, and can also be used by a user to change their own password. However, it is recommended that you use the CREATE USER statement to create users instead of the GRANT CONNECT statement.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

ALTER VIEW statement

Replaces a view definition with a modified version.

Syntax

- **Change the definition of a view**

ALTER VIEW

```
[ owner.]view-name [ ( column-name, ... ) ] AS query-expression  
[ WITH CHECK OPTION ]
```

- **Change the attributes of a view**

ALTER VIEW

```
[ owner.]view-name { SET HIDDEN | RECOMPILE | DISABLE | ENABLE }
```

Parameters

AS clause The SELECT statement on which the view is based. The SELECT statement must not refer to local temporary tables. Also, *query-expression* can have a GROUP BY, HAVING, WINDOW, or ORDER BY clause, and can contain UNION, EXCEPT, INTERSECT, or a common table expression.

Query semantics dictate that the order of the rows returned is undefined unless the query combines an ORDER BY clause with a TOP or FIRST clause in the SELECT statement.

WITH CHECK OPTION clause 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 *query-expression*.

SET HIDDEN clause Use the SET HIDDEN clause to obfuscate the definition of the view and cause the view to become hidden from view, for example in SQL Central. Explicit references to the view still work.

Note The SET HIDDEN operation is irreversible.

RECOMPILE clause Use the RECOMPILE clause to re-create the column definitions for the view. This clause is identical in functionality to the ENABLE clause, except that it can be used on a view that is not disabled. When a view is recompiled, the database server restores the column privileges based on the column names specified in the new view definition. The existing privileges are lost when a column no longer exists after the recompilation.

DISABLE clause Use the DISABLE clause to disable the view from use by the database server.

ENABLE clause Use the ENABLE clause to enable a disabled view. Enabling the view causes the database server to re-create the column definitions for the view. Before you enable a view, you must enable any views upon which it depends.

Remarks

The *query-expression* can specify a TOP n, FIRST, or LIMIT clause even if there is no ORDER BY clause. At most the specified number of rows are returned, but the order of the rows returned is not defined, so an ORDER BY could be specified but it is not required. When a view is used in a query, the ORDER BY in the view does not determine the order of rows in the query, even if there are no other tables in the FROM clause. That means that an ORDER BY should only be included in a view if

it is needed to select which rows are included by a TOP n, FIRST, or LIMIT clause. Otherwise, the ORDER BY clause has no effect and it is ignored by the database server.

If you execute an ALTER VIEW statement on a view that has one or more INSTEAD OF triggers, an error is returned. You must drop the trigger before the view can be dropped or altered.

If you alter a view owned by another user, you must qualify the name by including the owner (for example, GROUPO.ViewSalesOrders). If you don't qualify the name, the database server looks for a view with that name owned by you and alters it. If there isn't one, it returns an error.

When you alter a view, existing privileges on the view are maintained, and do not have to be reassigned. Instead of using the ALTER VIEW statement, you could also drop the view and recreate it using the DROP VIEW and CREATE VIEW, respectively. However, if you do so, privileges on the view need to be reassigned.

A query can specify a TOP n, FIRST, or LIMIT clause even if there is no ORDER BY clause. At most the specified number of rows are returned, but the order of the rows returned is not defined, so an ORDER BY could be specified but it is not required. When a view is used in a query, the ORDER BY in the view does not determine the order of rows in the query, even if there are no other tables in the FROM clause. That means that an ORDER BY should only be included in a view if it is needed to select which rows are included by a TOP n, FIRST, or LIMIT clause. Otherwise, the ORDER BY clause has no effect and it is ignored by the database server.

After completing the view alteration using the syntax to change the definition of a view, the database server recompiles the view. Depending on the type of change you made, if there are dependent views, the database server attempts to recompile them as well. If you have made a change that impacts a dependent view, you may need to alter the definition for the dependent view as well.

Caution

If the SELECT statement defining the view contained an asterisk (*), the number of the columns in the view may change if columns have been added or deleted from the underlying tables. The names and data types of the view columns may also change.

- **Change the definition of a view** This syntax is used to alter the structure of the view. Unlike altering tables where your change may be limited to individual columns, altering the structure of a view requires you to replace the entire view definition with a new definition, much as you would for creating the view.
- **Change the attributes of a view** This syntax is used to change attributes for the view, such as whether the view definition is hidden.

When you use SET HIDDEN, the view can be unloaded and reloaded into other databases. If SET HIDDEN is used, debugging using the debugger does not show the view definition, nor is it be available through procedure profiling. If you need to change the definition of a hidden view, you must drop the view and create it again using the CREATE VIEW statement.

When you use the DISABLE clause, the view is no longer available for use by the database server for answering queries. Disabling a view is similar to dropping it, except that the view definition remains in the database. Disabling a view also disables any dependent views. Therefore, the DISABLE clause requires exclusive access not only to the view being disabled, but also any dependent views, since they are disabled too.

Privileges

You must be the owner of the view, or have one of the following privileges:

- ALTER privilege on the view
- ALTER ANY VIEW system privilege
- ALTER ANY OBJECT system privilege

You must also have permission to select from the underlying objects for the view.

Side effects

Automatic commit.

All procedures and triggers are unloaded from memory, so that any procedure or trigger that references the view reflects the new view definition. The unloading and loading of procedures and triggers can have a performance impact if you are regularly altering views.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Related Information

[View dependencies](#)

A view definition refers to other objects such as columns, tables, and other views, and these references make the view **dependent** on the objects to which it refers.

[Views](#)

A view is a computed table that is defined by the result set of its view definition, which is expressed as a SQL query.

[Hiding the contents of a procedure, function, trigger, event, or view](#)

Use the SET HIDDEN clause to obscure the contents of a procedure, function, trigger, event, or view.

[Creating a regular view](#)

Create a view that combines data from one or more sources.

[CREATE VIEW statement](#)

Creates a view on the database.

[DROP VIEW statement](#)

Removes a view from the database.

[CREATE MATERIALIZED VIEW statement](#)

Creates a materialized view.

[ALTER MATERIALIZED VIEW statement](#)

Alters a materialized view.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

ATTACH TRACING statement (deprecated)

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database. The ATTACH TRACING statement starts a diagnostic tracing session (starts sending diagnostic information to the diagnostic tables).

Syntax

```
ATTACH TRACING TO { LOCAL DATABASE | connect-string }  

[ LIMIT { size | history } ]
```

connect-string : the connection string for the database

size : **SIZE** *nnn* { **MB** | **GB** }

history : **HISTORY** *nnn* { **MINUTES** | **HOURS** | **DAYS** }

nnn : integer

Parameters

connect-string The connection string is required to connect to the database receiving the tracing information. This parameter is only required when the database being profiled is different from the database receiving the data.

The following connection parameters are allowed in *connect-string*: DBF, DBKEY, DBN, Host, Server, LINKS, PWD, UID.

Specify the DBF relative to the database server that you want to connect to. If you do not specify a different database server, then the database server to which you are currently connected attempts to start the tracing database identified by the DBF connection parameter.

An error is returned if you specify the DBF parameter with the LINKS or Server connection parameters.

LIMIT clause The volume limit of data stored in the tracing database, either by size, or by length of time.

Remarks

The ATTACH TRACING statement is used to start a tracing session for the database you want to profile. You can only use it once a tracing level has been set. You can set the tracing level using the `sa_set_tracing_level` system procedure.

Once a session is started, tracing information is generated according to the tracing levels set in the `sa_diagnostic_tracing_level` table. You can send the tracing data to tracing tables within the same database that is being profiled by specifying LOCAL DATABASE. Alternatively, you can send the tracing data to a separate tracing database by specifying a connection string (*connect-string*) to that database. The tracing database must already exist, and you must have permissions to access it.

You can limit the amount of tracing data to store using the LIMIT SIZE or LIMIT HISTORY clauses. Use the LIMIT SIZE clause when you want to limit the volume of tracing data to a certain size, as measured in megabytes or gigabytes. Use the LIMIT HISTORY clause to limit the volume of tracing data to a period of time, as measured in minutes, hours, or days. For example, `HISTORY 8 DAYS` limits

the amount of tracing data stored in the tracing database to 8 days' worth.

To start a tracing session, TCP/IP must be running on the database server(s) on which the tracing database and production database are running.

Packets that contain potentially sensitive data are visible on the network interface, even when tracing to a local database. For security purposes, you can specify encryption in the connection string.

To see the current tracing levels set for a database, look in the `sa_diagnostic_tracing_level` table.

To see where tracing data is being sent to, examine the `SendingTracingTo` database property.

You must be connected to the database being profiled to execute this statement.

Privileges

You must have the `DIAGNOSTICS` system role and the `MANAGE PROFILING` system privilege.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example sets the tracing level to 1 using the `sa_set_tracing_level` system procedure. Then it starts a tracing session. Tracing data generated for the local database will be sent to the `mytracingdb` tracing database on another computer, as shown by the specified connection string. A maximum of two hours of tracing data will be maintained during the tracing session.

```
CALL sa_set_tracing_level( 1 );  
ATTACH TRACING TO 'UID=DBA;PWD=passwd;Server=remotedbsrv;DBN=mytracingdb;  
Host=mytracing-pc'  
LIMIT HISTORY 2 HOURS;
```

Related Information

[Troubleshoot and fine tune applications with the SQL Anywhere Profiler](#)

Use the Profiler to quickly view SQL statements that are currently running in your database, compare run times of procedures, and analyze how the objects in your database interact with each other.

[Connection parameters](#)

Connection parameters are included in connection strings.

[TCP/IP protocol](#)

TCP/IP is used to connect clients to databases running on different computers.

[Diagnostic tracing \(deprecated\)](#)

The Diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database.

[DETACH TRACING statement \(deprecated\)](#)

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database. The `DETACH TRACING` session ends a diagnostic tracing session.

[REFRESH TRACING LEVEL statement \(deprecated\)](#)

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in

your database. The REFRESH TRACING LEVEL statement reloads the tracing levels from the sa_diagnostic_tracing_level table while a tracing session is in progress.

[-x database server option](#)

Specifies server-side network communications protocols.

[sa_diagnostic_tracing_level table \(deprecated\)](#)

The sa_diagnostic_tracing_level table is owned by the dbo user, and each row in this table is a condition that determines what kind of diagnostic information to send to the tracing database.

[sa_set_tracing_level system procedure \(deprecated\)](#)

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database. Initializes the level of tracing information to be stored in the diagnostic tracing tables.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

BACKUP DATABASE statement

Backs up a database and transaction log.

Syntax

- **Image backup**

BACKUP DATABASE

DIRECTORY *backup-directory*

[*backup-option* [*backup-option ...*]]

backup-directory : { *string* | *variable* }

backup-option :

WAIT BEFORE START

| **WAIT AFTER END**

| **DBFILE ONLY**

| **TRANSACTION LOG ONLY**

| **TRANSACTION LOG RENAME** [**MATCH**]

| **TRANSACTION LOG TRUNCATE**

| **ON EXISTING ERROR**

| **WITH COMMENT** *comment-string*

| **HISTORY** { **ON** | **OFF** }

| **AUTO TUNE WRITERS** { **ON** | **OFF** }

| **WITH CHECKPOINT LOG** { **AUTO** | **COPY** | **NO COPY** | **RECOVER** }

- **Archive backup**

BACKUP DATABASE TO *archive-root*

[*backup-option* [*backup-option ...*]]

archive-root : { *string* | *variable* }

backup-option :

WAIT BEFORE START

| **WAIT AFTER END**

| **DBFILE ONLY**

| **TRANSACTION LOG ONLY**

| **TRANSACTION LOG RENAME** [**MATCH**]

| **TRANSACTION LOG TRUNCATE**

| **ATTENDED** { **ON** | **OFF** }

| **WITH COMMENT** *comment-string*

| **HISTORY** { **ON** | **OFF** }

| **WITH CHECKPOINT LOG** [**NO**] **COPY**

| **MAX WRITE** { *number-of-writers* | **AUTO** }

| **FREE PAGE ELIMINATION** { **ON** | **OFF** }

comment-string : *string*

number-of-writers : *integer*

Parameters

DIRECTORY clause The target location on disk for the backup files, relative to the database server's current directory at startup. If the directory does not exist, it is created. Specifying an empty string as a directory allows you to rename or truncate the transaction log without first making a copy of it. Do not use this clause if you are using database mirroring.

WAIT BEFORE START clause This clause delays the backup until there are no active transactions. All other activity on the database is prevented and a checkpoint is performed.

Using this clause with the WITH CHECKPOINT LOG NO COPY clause verifies that the backup copy of the database does not require recovery and allows you to start the backup copy of the database in read-only mode and validate it. When you validate the backup database, you do not need to make an additional copy of the database.

WAIT AFTER END clause This clause ensures that all transactions are completed before the transaction log is renamed or truncated. The database server waits for other connections to commit or rollback any open transactions before finishing the backup. Use this clause with caution as new, incoming transactions can cause the backup to wait indefinitely.

DBFILE ONLY clause This clause makes backup copies of the main database file and all associated dbspaces, but not the transaction log. You cannot use the DBFILE ONLY clause with the TRANSACTION LOG RENAME or TRANSACTION LOG TRUNCATE clauses.

TRANSACTION LOG ONLY clause You can specify the TRANSACTION LOG ONLY clause to create a backup copy of the transaction log, without copying the other database files.

TRANSACTION LOG RENAME [MATCH] clause This clause causes the database server to rename the current transaction log to a file name of the form *YYMMDDnn.log* and start a *new* transaction log that has the same name as the database file. The backup transaction log gets the same name as the active transaction log unless MATCH is specified. If MATCH is specified, the backup copy of the transaction log gets the same name as the renamed file (*YYMMDDnn.log*). Using the MATCH keyword enables the same statement to be executed several times without writing over old data.

The transaction log can be renamed and restarted without completing a backup by specifying an empty directory name with the TRANSACTION LOG ONLY clause. For example:

```
BACKUP DATABASE DIRECTORY ''  
TRANSACTION LOG ONLY  
TRANSACTION LOG RENAME;
```

TRANSACTION LOG TRUNCATE clause If this clause is used, the current transaction log is truncated and restarted at the completion of the backup. Do not use this clause if you are using database mirroring.

The transaction log can be truncated without completing a backup by specifying an empty directory name with the TRANSACTION LOG ONLY clause. For example:

```
BACKUP DATABASE DIRECTORY ''  
TRANSACTION LOG ONLY  
TRANSACTION LOG TRUNCATE;
```

archive-root clause The file name or tape drive device name for the archive file.

To back up to tape, you must specify the device name of the tape drive. The number automatically appended to the end of the archive file name is incremented each time you

execute an archive backup.

The backslash (\) is an escape character in SQL strings, so each backslash must be doubled.

ON EXISTING ERROR clause This clause applies only to image backups. By default, existing files are overwritten when you execute a BACKUP DATABASE statement. If this clause is used, an error occurs if any of the files to be created by the backup already exist.

ATTENDED clause The clause applies only when backing up to a tape device. ATTENDED ON (the default) indicates that someone is available to monitor the status of the tape drive and to place a new tape in the drive when needed. A message is sent to the application that issued the BACKUP DATABASE statement if the tape drive requires intervention. The database server then waits for the drive to become ready. This may happen, for example, when a new tape is required.

If ATTENDED OFF is specified and a new tape is required or the drive is not ready, no message is sent and an error is given.

WITH COMMENT clause This clause records a comment in the backup history file. For archive backups, the comment is also recorded in the archive file.

HISTORY clause This clause enables or disables backup history. By default, this clause is ON, meaning that each backup operation appends a line to the *backup.syb* file. Specifying HISTORY OFF prevents updates to the *backup.syb* file, and is recommended when:

- The database is backed up frequently.
- There is no procedure in place to periodically archive or delete the *backup.syb* file.
- Disk space is limited.

AUTO TUNE WRITERS clause Specifying this clause enables or disables the automatic tuning of writers. During the backup process, one writer writes the backup files to the backup directory. If the backup directory is on a device that can handle an increased writer load (such as a RAID array), the default AUTO TUNE WRITERS ON improves overall backup performance by increasing the number of writers. The database server periodically examines the read and write performances of all devices that are participating in the backup. Specifying AUTO TUNE WRITERS OFF prevents the database server from creating additional writers.

WITH CHECKPOINT LOG clause This clause specifies how the backup processes the database files before writing them to the destination directory. The choice of whether to apply pre-images during a backup, or copy the checkpoint log as part of the backup, has performance implications. The default setting is AUTO for image backups and COPY for archive backups.

- **COPY clause** This option cannot be used with the WAIT BEFORE START clause of the BACKUP DATABASE statement.

When you specify COPY, the backup reads the database files without applying any modified pages. The entire checkpoint log and the system dbspace are copied to the backup directory. The next time the database server is started, the database server automatically recovers the database to the state it was in as of the checkpoint at the time the backup started.

Because pages do not have to be written to the temporary file, using this option can provide better backup performance, and reduce internal server contention for other connections that are operating during a backup. However, since the checkpoint log contains original images of modified pages, it grows in the presence of database updates. With copy specified, the backed-up copy of the database files may be larger than the database files at the time the backup started. The COPY option should be used if disk space in the destination directory is not an issue.

- **NO COPY clause** When you specify NO COPY, the checkpoint log is not copied as part of the backup. This option causes modified pages to be saved in the temporary file so that they can be applied to the backup as it progresses. The backup copies of the database files are the same size as the database when the backup operation commenced.

This option results in smaller backed up database files, but the backup may proceed more slowly, and possibly decrease performance of other operations in the database server. It is useful in situations where space on the destination drive is limited.

- **RECOVER clause** When you specify RECOVER, the database server copies the checkpoint log (as with the COPY option), but applies the checkpoint log to the database when the backup is complete. This restores the backed up database files to the same state (and size) that they were in at the start of the backup operation. This option is useful if space on the backup drive is limited (it requires the same amount of space as the COPY option for backing up the checkpoint log, but the resulting file size is smaller).
- **AUTO clause** When you specify AUTO, the database server checks the amount of available disk space on the volume hosting the backup directory. If there is at least twice as much disk space available as the size of the database at the start of the backup, then this option behaves as if copy were specified. Otherwise, it behaves as NO COPY. AUTO is the default behavior.

MAX WRITE clause For archive backups, by default one thread is dedicated to writing the backup files. If the backup directory is on a device that can handle an increased writer load (such as a RAID array), then overall backup performance can be improved by increasing the number of threads acting as writers.

If AUTO is specified, one output stream is created for each reader thread. The value *n* specifies the maximum number of output streams that can be created, up to the number of reader threads. The default value for this clause is 1. If you are backing up to tape, only one writer can be used.

The first stream, stream 0, produces files named *myarchive.X*, where *X* is a number that starts at 1 and continues incrementing to the number of files required. All of the other streams produce files named *myarchive.YZ*, where *Y* is the stream number (starting at 1), and *Z* is a number that starts at 1 and continues incrementing to the number of files required.

FREE PAGE ELIMINATION clause By default, archive backups skip some free pages, which can result in smaller and potentially faster backups. Free page elimination has no effect on the back up of transaction log files because transaction log files do not contain free pages. Databases with large transaction log files may not benefit as much from free page elimination as databases with small transaction log files.

When you back up a strongly encrypted database with free page elimination turned on, you must specify the encryption key when restoring the database. When you back up a strongly encrypted database with free page elimination turned off, you do not need to specify the encryption key when restoring the database.

As of version 12, you cannot restore archive backups created with version 11 or earlier database servers.

Remarks

The BACKUP DATABASE statement performs a server-side backup. To perform a client-side backup, use the dbbackup utility.

If the disk sandbox feature is enabled for the database, you must specify a secure feature key that

disables the disk sandbox feature for the database server to be able to make the backup in a directory outside of the sandbox (the directory where the main database file is located and any subdirectories of this directory).

Each backup operation, whether image or archive, updates a history file called *backup.syb*. This file records the BACKUP and RESTORE operations that have been performed on a database server.

To create a backup that can be started on a read-only server without having to go through recovery, you must use both the WAIT BEFORE START and WITH CHECKPOINT LOG NO COPY clauses. The WAIT BEFORE START clause ensures that the rollback log is empty, and the WITH CHECKPOINT LOG NO COPY clause ensures that the checkpoint log is empty. If either of these files is missing, then recovery is required. You can use WITH CHECKPOINT LOG RECOVER as an alternative to the WAIT BEFORE START and WITH CHECKPOINT LOG NO COPY clauses if you do not need to recover the database you backed up.

- **Syntax - Image backup** An image backup creates copies of each of the database files, in the same way as the Backup utility (dbbackup). By default, the Backup utility makes the backup on the client computer, but you can specify the -s option to create the backup on the database server when using the Backup utility. For the BACKUP DATABASE statement, however, the backup can only be made on the database server.

Optionally, only the database file(s) or transaction log can be saved. The transaction log may also be renamed or truncated after the backup has completed.

Alternatively, you can specify an empty string as a directory to rename or truncate the log without copying it first. This is useful in a replication environment where space is a concern. You can use this feature with an event handler on transaction log size to rename the transaction log when it reaches a given size, and with the delete_old_logs option to delete the transaction log when it is no longer needed.

To restore from an image backup, copy the saved files back to their original locations and reapply the transaction logs.

- **Syntax - Archive backup** An archive backup creates a single file holding all the required backup information. The destination can be either a file name or a tape drive device name.

There can be only one backup on a given tape. The tape is ejected at the end of the backup.

Only one archive per tape is allowed, but a single archive can span multiple tapes. To restore a database from an archive backup, use the RESTORE DATABASE statement.

If a RESTORE DATABASE statement references an archive file containing only a transaction log, the statement must specify a file name for the location of the restored database file, even if that file does not exist. For example, to restore from an archive that only contains a transaction log to the directory C:\MYNEWDB, the RESTORE DATABASE statement is:

```
RESTORE DATABASE 'c:\\temp\\mynewdb\\my.db' FROM archive-root
```

Caution

Backup copies of the database and transaction log must not be changed in any way. If there were no transactions in progress during the backup, or if you specified BACKUP DATABASE WITH CHECKPOINT LOG RECOVER or WITH CHECKPOINT LOG NO COPY, you can check the validity of the backup database using read-only mode or by validating a copy of the backup database.

However, if transactions were in progress, or if you specified BACKUP DATABASE

WITH CHECKPOINT LOG COPY, the database server must perform recovery on the database when you start it. Recovery modifies the backup copy, which is not desirable.

During the execution of this statement, you can request progress messages.

You can also use the Progress connection property to determine how much of the statement has been executed.

Privileges

You must have the BACKUP DATABASE system privilege.

Side effects

Causes a checkpoint.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

Back up the current database and the transaction log, each to a different file, and rename the existing transaction log. An image backup is created.

```
BACKUP DATABASE  
DIRECTORY 'c:\\temp\\backup'  
TRANSACTION LOG RENAME;
```

The option to rename the transaction log is useful, especially in replication environments where the old transaction log is still required.

Back up the current database and transaction log to tape:

```
BACKUP DATABASE  
TO '\\\\.\\tape0';
```

Rename the transaction log without making a copy:

```
BACKUP DATABASE DIRECTORY ''  
TRANSACTION LOG ONLY  
TRANSACTION LOG RENAME;
```

Execute the BACKUP DATABASE statement with a dynamically constructed directory name:

```
CREATE EVENT NightlyBackup
SCHEDULE
START TIME '23:00' EVERY 24 HOURS
HANDLER
BEGIN
    DECLARE dest LONG VARCHAR;
    DECLARE day_name CHAR(20);

    SET day_name = DATENAME( WEEKDAY, CURRENT DATE );
    SET dest = 'd:\\backups\\' || day_name;
        BACKUP DATABASE DIRECTORY dest
    TRANSACTION LOG RENAME;
END;
```

Related Information

[Database backup and recovery](#)

Use backups to restore all committed changes to a database up to the time it became unavailable.

[Types of backup](#)

There are many choices to make when deciding when, where, and how to perform a backup. The decisions you make define the backup type to perform (for example, server-side and client-side, full and incremental).

[Transaction log file management in a database mirroring system](#)

When a partner server starts, it examines all the transaction log files in the same directory as the current transaction log file and determines which ones must be applied.

[Backing up a database using the BACKUP DATABASE statement](#)

Backup a database using the BACKUP DATABASE statement.

[progress_messages option](#)

Controls whether progress messages are sent from the database server to the client.

[RESTORE DATABASE statement](#)

Restores a backed up database from an archive.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

BEGIN SNAPSHOT statement

Starts a snapshot at a specified period in time for use with snapshot isolation transactions.

Syntax

BEGIN SNAPSHOT

Remarks

By default, when a transaction begins, the database server defers creating the snapshot until the application causes the first row of a table to be fetched. You can use the BEGIN SNAPSHOT statement to start the snapshot earlier within the transaction. The database server creates a snapshot when the BEGIN SNAPSHOT statement is executed by a snapshot transaction.

The statement fails and returns an error when either of the following conditions is met:

- support for snapshots transactions has not been enabled for the database.
- a snapshot has already been started for the current transaction.

This statement is also useful for non-snapshot transactions because it allows them to start a snapshot that can be used later in the transaction for a statement-level snapshot operation.

Privileges

None.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Related Information

[Snapshot isolation](#)

Snapshot isolation is designed to improve concurrency and consistency by maintaining different versions of data.

[allow_snapshot_isolation option](#)

Controls whether snapshot isolation is enabled or disabled.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

BEGIN statement

Specifies a compound statement.

Syntax

```
[ statement-label : ]  
BEGIN [ [ NOT ] ATOMIC ]  
  [ local-declaration; ... ]  
  statement-list  
  [ EXCEPTION [ exception-case ... ] ]  
END [ statement-label ]
```

local-declaration :
 variable-declaration
 | *cursor-declaration*
 | *exception-declaration*
 | *temporary-table-declaration*

exception-case :
 WHEN *exception-name* [, ...] **THEN** *statement-list*
 | **WHEN OTHERS** **THEN** *statement-list*

variable-declaration and *exception-declaration* : [see the DECLARE statement](#)

cursor-declaration : [see the DECLARE CURSOR statement](#)

temporary-table-declaration : [see the DECLARE LOCAL TEMPORARY TABLE statement](#)

Parameters

statement-label 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 trigger has an implicit label that is the same as the name of the procedure or trigger.

ATOMIC clause An atomic statement is a statement that is executed completely or not at all. For example, an UPDATE statement that updates thousands of rows might encounter an error after updating many rows. If the statement does not complete, all changes revert back to their original state. Similarly, if you specify that the BEGIN statement is atomic, the statement is executed either in its entirety or not at all.

local-declaration Immediately following the BEGIN, a compound statement can have local declarations for objects 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 of the compound statement are visible to the exception handlers for the statement. Local declarations are not visible to other procedures that are called from within a compound statement.

Remarks

The body of a procedure or trigger is a compound statement. 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.

Privileges

None.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** BEGIN, which identifies a compound statement, comprises part of optional ANSI/ISO SQL Language Feature P002.

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 CompanyName, CAST(
      sum( SalesOrderItems.Quantity *
        Products.UnitPrice ) AS INTEGER) VALUE
    FROM GROUP0.Customers
      LEFT OUTER JOIN SalesOrders
      LEFT OUTER JOIN SalesOrderItems
      LEFT OUTER JOIN Products
    GROUP BY CompanyName;
  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;
```

The example below declares the following variables:

- v1 as an INT with the initial setting of 5.
- v2 and v3 as CHAR(10), both with an initial value of abc.

```
BEGIN
  DECLARE v1 INT = 5
  DECLARE v2, v3 CHAR(10) = 'abc'
  // ...
END
```

Related Information

[Stored procedures, triggers, batches, and user-defined functions](#)

Procedures and triggers store procedural SQL statements in a database.

[Error and warning handling](#)

After an application program executes a SQL statement, it can examine a **status code** (or return code) which indicates whether the statement executed successfully or failed and gives the reason for the failure.

[Exception handling and atomic compound statements](#)

If an error occurs within an atomic compound statement and that statement has an exception handler that handles the error, then the compound statement completes without an active exception and the changes before the exception are not reversed.

[Exception handlers](#)

You can intercept certain types of errors and handle them within a procedure or trigger, rather than pass the error back to the calling environment. This is done through the use of an **exception handler**.

[Atomic compound statements](#)

An **atomic** statement is a statement that is executed completely or not at all.

[DECLARE statement](#)

Declares a SQL variable (connection-scope) or an exception within a compound statement (BEGIN...END).

[DECLARE CURSOR statement \[ESQL\] \[SP\]](#)

Declares a cursor.

[DECLARE LOCAL TEMPORARY TABLE statement](#)

Declares a local temporary table.

[CONTINUE statement](#)

Restarts a loop.

[SIGNAL statement \[SP\]](#)

Signals an exception condition.

[RESIGNAL statement \[SP\]](#)

Resignals an exception condition.

[RAISERROR statement](#)

Signals an error and sends a message to the client.

[BEGIN statement \[TSQL\]](#)

Specifies a compound statement.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

BEGIN statement [TSQL]

Specifies a compound statement.

Syntax

BEGIN

statement-list

END

statement-list :

sql-statement

| *variable-declaration*

| *cursor-declaration*

| *temporary-table-declaration*

variable-declaration : [see the DECLARE statement](#)

cursor-declaration : [see the DECLARE CURSOR statement](#)

temporary-table-declaration : [see the DECLARE LOCAL TEMPORARY TABLE statement](#)

Parameters

statement-list A list of statements and declarations.

Remarks

A BEGIN statement allows one or more SQL statements to be grouped together and treated as a unit, and starts with the keyword BEGIN and ends with the keyword END.

Error handling is different for Transact-SQL compound statements.

Privileges

None.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** BEGIN, which identifies a compound statement, comprises part of optional ANSI/ISO SQL Language Feature P002.

Example

The example below declares the following variables:

- v1 as an INT with the initial setting of 5.
- v2 and v3 as CHAR(10), both with an initial value of abc.


```
BEGIN
  DECLARE v1 INT = 5
  DECLARE v2, v3 CHAR(10) = 'abc'
          // ...
END
```

Related Information

[Error handling in Transact-SQL procedures](#)

Default procedure error handling is different in the Watcom SQL and Transact-SQL dialects.

[Stored procedures, triggers, batches, and user-defined functions](#)

Procedures and triggers store procedural SQL statements in a database.

[Transact-SQL-compatible databases](#)

You can eliminate some differences in behavior between SQL Anywhere and Adaptive Server Enterprise by selecting appropriate options when creating a database or when rebuilding an existing database.

[DECLARE statement](#)

Declares a SQL variable (connection-scope) or an exception within a compound statement (BEGIN...END).

[DECLARE CURSOR statement \[ESQL\] \[SP\]](#)

Declares a cursor.

[DECLARE LOCAL TEMPORARY TABLE statement](#)

Declares a local temporary table.

[BEGIN statement](#)

Specifies a compound statement.

[CONTINUE statement](#)

Restarts a loop.

[GOTO statement](#)

Branches to a labeled statement.

[RAISERROR statement](#)

Signals an error and sends a message to the client.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

BEGIN TRANSACTION statement [T-SQL]

Begins a user-defined transaction.

Syntax

BEGIN TRAN[SACTION] [*transaction-name*]

Remarks

The optional parameter *transaction-name* is the name assigned to this transaction. It must be a valid identifier. Use transaction names only on the outermost pair of nested BEGIN/COMMIT or BEGIN/ROLLBACK statements.

When executed inside a transaction, the BEGIN TRANSACTION statement increases the nesting level of transactions by one. The nesting level is decreased by a COMMIT statement. When transactions are nested, only the outermost COMMIT makes the changes to the database permanent.

Both Adaptive Server Enterprise and SQL Anywhere have two transaction modes.

The default Adaptive Server Enterprise transaction mode, called unchained mode, commits each statement individually, unless an explicit BEGIN TRANSACTION statement is executed to start a transaction. In contrast, the ANSI/ISO SQL Standard compatible chained mode only commits a transaction when an explicit COMMIT is executed or when a statement that carries out an autocommit (such as a data definition statement) is executed.

You can control the mode by setting the chained database option. The default setting for ODBC and Embedded SQL connections in SQL Anywhere is On, in which case the database server runs in chained mode. (ODBC users should also check the AutoCommit ODBC setting). The default for TDS connections is Off.

In unchained mode, a transaction is implicitly started before any data retrieval or manipulation statement. These statements include: DELETE, INSERT, OPEN, FETCH, SELECT, and UPDATE. You must still explicitly end the transaction with a COMMIT or ROLLBACK statement.

You cannot alter the setting of the chained option within a transaction.

Caution

When calling a stored procedure, you should ensure that it operates correctly under the required transaction mode.

The current nesting level is held in the global variable @@trancount. The @@trancount variable has a value of zero before the first BEGIN TRANSACTION statement is executed, and only a COMMIT executed when @@trancount is equal to one makes changes to the database permanent.

You should not rely on the value of @@trancount for more than keeping track of the number of explicit BEGIN TRANSACTION statements that have been executed.

When Adaptive Server Enterprise starts a transaction implicitly, the @@trancount variable is set to 1. SQL Anywhere does not set the @@trancount value to 1 when a transaction is started implicitly. Instead, the SQL Anywhere @@trancount variable has a value of zero before any BEGIN TRANSACTION statement (even though there is a current transaction), while in Adaptive Server Enterprise (in chained mode) it has a value of 1.

For transactions starting with a BEGIN TRANSACTION statement, @@trancount has a value of 1 in both SQL Anywhere and Adaptive Server Enterprise after the first BEGIN TRANSACTION statement. If a transaction is implicitly started with a different statement, and a BEGIN TRANSACTION statement is then executed, @@trancount has a value of 2 in both SQL Anywhere, and Adaptive Server Enterprise after the BEGIN TRANSACTION statement.

A ROLLBACK statement without a transaction or savepoint name always rolls back statements to the outermost BEGIN TRANSACTION (explicit or implicit) statement, and cancels the entire transaction.

Privileges

None.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.
- **Transact-SQL** BEGIN TRANSACTION is supported by Adaptive Server Enterprise.

Example

The following batch reports successive values of @@trancount as 0, 1, 2, 1, and 0. The values are printed in the database server messages window.

```
PRINT @@trancount
BEGIN TRANSACTION
PRINT @@trancount
BEGIN TRANSACTION
PRINT @@trancount
COMMIT
PRINT @@trancount
COMMIT
PRINT @@trancount
```

Related Information

[Savepoints within transactions](#)

You can define **savepoints** in a transaction to separate groups of related statements.

[COMMIT statement](#)

Makes changes to the database permanent, or terminates a user-defined transaction.

[ROLLBACK statement](#)

Ends a transaction and undo any changes made since the last COMMIT or ROLLBACK.

[SAVEPOINT statement](#)

Establishes a savepoint within the current transaction.

[isolation_level option](#)

Controls the locking isolation level.

[chained option](#)

Controls the transaction mode in the absence of a BEGIN TRANSACTION statement.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

BREAK statement [T-SQL]

Exits a compound statement or loop.

Syntax

BREAK

Remarks

The BREAK statement is a control statement that allows you to leave a loop. Execution resumes at the first statement after the loop.

Privileges

None.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

In this example, the BREAK statement breaks the WHILE loop if the most expensive product has a price above \$50. Otherwise, the loop continues until the average price is greater than or equal to \$30:

```
WHILE ( SELECT AVG( UnitPrice ) FROM Products ) < $30
BEGIN
    UPDATE GROUP0.Products
    SET UnitPrice = UnitPrice + 2
    IF ( SELECT MAX(UnitPrice) FROM Products ) > $50
        BREAK
END
```

Related Information

[Stored procedures, triggers, batches, and user-defined functions](#)

Procedures and triggers store procedural SQL statements in a database.

[WHILE statement \[T-SQL\]](#)

Provides repeated execution of a statement or compound statement.

[CONTINUE statement](#)

Restarts a loop.

[BEGIN statement](#)

Specifies a compound statement.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

CALL statement

Invokes a procedure.

Syntax

- **Specify argument by position**

```
[variable = ] CALL procedure-name ( [ expression, ... ] ) [ AS USER { string | variable } IDENTIFIED BY { string | variable } ]
```

- **Specify argument by keyword format**

```
[variable = ] CALL procedure-name ( [ parameter-name = expression, ... ] ) [ AS USER { string | variable } IDENTIFIED BY { string | variable } ]
```

Parameters

- **AS USER ... IDENTIFIED BY clause** This optional clause calls a procedure or function as a different user. The database server verifies that the user ID and password provided are valid, and then executes the procedure or function as the specified user. The invoker of the procedure is the specified user. Upon exiting the procedure or function, the user context is restored to its original state.

Note All string values must be enclosed in single quotes; otherwise the database server interprets them as variable names.

Remarks

The CALL statement invokes a procedure that was previously created with a CREATE PROCEDURE statement. When the procedure completes, any INOUT or OUT parameter value is copied back.

Note The AS USER ... IDENTIFIED BY clause only applies to the CALL statement and is not supported for procedures in the FROM clause or functions in the select list.

If you use this statement in a procedure, do not specify the password (IDENTIFIED BY clause) as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

Database-scope variables must not be specified as INOUT or OUT parameters when calling a procedure.

The argument list can be specified by position or by using keyword format. By position, the arguments match up with the corresponding parameter in the parameter list for the procedure (DEFAULT can be used for an optional parameter). By keyword, the arguments are matched up with the named parameters.

Procedure arguments can be assigned default values in the CREATE PROCEDURE statement, and missing parameters are assigned the default value. If no default is set, and an argument is not provided, an error is given.

Inside a procedure, a CALL statement can be used in a DECLARE statement when the procedure returns result sets.

Subqueries and spatial method calls are not allowed as arguments to a stored procedure in a CALL statement.

Procedures can return an integer value (for example, as a status indicator) using the RETURN statement. You can save this return value in a variable using the equality sign as an assignment operator:

If the procedure being called returns an INT and the value is NULL, then the error status value, 0, is returned instead. There is no way to differentiate between this case and the case of an actual value of 0 being returned.

Note

Use of this statement to invoke a function is deprecated. If you have a function you want to call, consider using an assignment statement to invoke the function and assign its result to a variable. For example:

```
DECLARE varname INT;  
SET varname=test( );
```

Privileges

The user calling the procedure, or the user specified by the AS USER ... IDENTIFIED BY clause, must be the owner of the procedure, or have one of the following privileges:

- EXECUTE privilege on the procedure
- EXECUTE ANY PROCEDURE system privilege

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Core Feature. The use of the RETURN statement to return a value from a stored procedure is not part of the standard; the ANSI/ISO SQL Standard supports return values only for SQL-invoked functions, not for procedures. Default values for stored procedure arguments are not in the standard.

Example

The following 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.

```

CREATE PROCEDURE OrderCount ( IN customer_ID INT, OUT Orders INT )
BEGIN
    SELECT COUNT( GROUP0.SalesOrders.ID )
    INTO Orders
    FROM GROUP0.Customers
    KEY LEFT OUTER JOIN SalesOrders
    WHERE Customers.ID = customer_ID;
END
go
-- Create a variable to hold the result
CREATE VARIABLE Orders INT
go
-- Call the procedure, FOR customer 101
CALL OrderCount ( 101, Orders )
go
-- Display the result
SELECT Orders FROM SYS.DUMMY
go

```

The following example uses the AS USER...IDENTIFIED BY clause to execute a procedure as a different user:

1. Create three sample users by executing the following statements:

```

CREATE USER u IDENTIFIED BY pwdforu;
CREATE USER u1 IDENTIFIED BY pwdforu1;
CREATE USER u2 IDENTIFIED BY pwdforu2;

```

2. Create and populate two tables by executing the following statements:

```

CREATE TABLE u1.t1 (c1 INT);
CREATE TABLE u2.t2 (c1 INT);
INSERT INTO u1.t1 VALUES(1);
INSERT INTO u2.t2 VALUES(2);
COMMIT;

```

3. Create two stored procedures by executing the following statements:

```

CREATE PROCEDURE u1.p1( OUT ret INT, OUT inv_user CHAR(128), OUT exec_user
CHAR(128) )
SQL SECURITY INVOKER
BEGIN
    SELECT c1 INTO ret FROM t1;
    SET inv_user = invoking_user;
    SET exec_user = executing_user;
END;

```



```

CREATE PROCEDURE u2.p2( OUT ret INT, OUT inv_user CHAR(128), OUT exec_user
CHAR(128) )
SQL SECURITY DEFINER
BEGIN
    SELECT c1 INTO ret FROM t2;
    SET inv_user = invoking_user;
    SET exec_user = executing_user;
END;

```

4. Create a third procedure that calls the first procedure as user u1, using string values in the AS USER...IDENTIFIED BY clause:

```

CREATE PROCEDURE u.p1( OUT ret INT, OUT inv_user CHAR(128), OUT exec_user
CHAR(128) )
SQL SECURITY DEFINER
BEGIN
    CALL u1.p1 ( ret, inv_user, exec_user ) AS USER 'u1' IDENTIFIED BY
'pwdforu1';
END;

```

5. Create a fourth procedure that calls the second procedure as user u, using variable values in the AS USER...IDENTIFIED BY clause:

```

CREATE PROCEDURE u.p2( IN u CHAR(128), IN p CHAR(128), OUT ret INT, OUT
inv_user CHAR(128), OUT exec_user CHAR (128) )
SQL SECURITY DEFINER
BEGIN
    CALL u2.p2( ret, inv_user, exec_user ) AS USER u IDENTIFIED BY p;
END;

```

6. Now, if user u logs in and executes CALL u1.p1(ret, inv_user, exec_user) OR CALL u2.p2(ret, inv_user, exec_user), then user u has permission denied since user u does not have permission to execute u1.p1 or u2.p2. However, user u is able to execute the following procedure:

```
CALL u1.p1( ret, inv_user, exec_user ) AS USER 'u1' IDENTIFIED BY 'pwdforu1';
```

Both the inv_user and exec_user are returned as u1 even though u1.p1 is a SQL SECURITY INVOKER procedure. This is because the database server executes u1.p1 as user u1, even though the user logged in is user u. Because the procedure executes as though u1 is calling it, it can access the non-fully qualified table t1, since u1.t1 exists.

Similarly, u is able to execute the following procedure:

```
CALL u2.p2( ret, inv_user, exec_user ) AS USER uvar IDENTIFIED BY pvar;
```

Here, uvar is a variable that contains the value 'u2' and pvar is a variable that contains the value 'pwdforu2'. The procedure executes without error and the inv_user and exec_user both come back as u2 and the procedure can access the non-fully qualified table t2.

7. If user u executes the following two procedures, then both calls succeed since the database server accepts the AS USER...IDENTIFIED BY clause nested within other stored procedures:

```
CALL u.p1( ret, inv_user, exec_user );
```

```
CALL u.p2( 'u2', 'pwdforu2', ret, inv_user, exec_user );
```

Related Information

[Result sets](#)

Procedures can return results in the form of a single row of data, or multiple rows.

[Stored procedures, triggers, batches, and user-defined functions](#)

Procedures and triggers store procedural SQL statements in a database.

[CREATE FUNCTION statement](#)

Creates a user-defined SQL function in the database.

[CREATE FUNCTION statement \[External call\]](#)

Creates an interface to a native or external function.

[CREATE FUNCTION statement \[Web service\]](#)

Creates a web client function that makes an HTTP or SOAP over HTTP request.

[CREATE PROCEDURE statement](#)

Creates a user-defined SQL procedure in the database.

[CREATE PROCEDURE statement \[External call\]](#)

Creates an interface to a native or external procedure.

[CREATE PROCEDURE statement \[Web service\]](#)

Creates a user-defined web client procedure that makes HTTP or SOAP requests to an HTTP server.

[EXECUTE statement \[T-SQL\]](#)

Invokes a procedure (an Adaptive Server Enterprise-compatible alternative to the CALL statement), executes a prepared SQL statement in Transact-SQL.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CASE statement

Selects an execution path based on multiple cases.

Syntax

- **Specify value expressions**

```
CASE value-expression  
WHEN [ constant | NULL ] THEN statement-list ...  
[ WHEN [ constant | NULL ] THEN statement-list ] ...  
[ ELSE statement-list ]  
END [ CASE ]
```

- **Specify search conditions**

```
CASE  
WHEN [ search-condition | NULL ] THEN statement-list ...  
[ WHEN [ search-condition | NULL ] THEN statement-list ] ...  
[ ELSE statement-list ]  
END [ CASE ]
```

Remarks

CASE statement using value expressions 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. The *value-expression* is an expression that takes on a single value, which may be a string, a number, a date, or other SQL data type. If a WHEN clause exists for the value of *value-expression*, the *statement-list* in the WHEN clause is executed. If no appropriate WHEN clause exists, and an ELSE clause exists, the *statement-list* in the ELSE clause is executed. Execution resumes at the first statement after the END CASE.

If the *value-expression* can be null, use the ISNULL function to replace the NULL *value-expression* with a different expression.

CASE statement using search conditions With this form, the statements are executed for the first satisfied *search-condition* in the CASE statement. The ELSE clause is executed if none of the *search-conditions* are met.

If the expression can be NULL, use the following syntax for the first *search-condition*:

```
WHEN search-condition IS NULL THEN statement-list
```

Note Do not confuse the syntax of the CASE statement with that of the CASE expression.

Privileges

None.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** The CASE statement is part of Language Feature P002 (Computational completeness). The use of END alone, rather than END CASE, is not in the standard.
- **Transact-SQL** The CASE statement is supported by Adaptive Server Enterprise.

Example

The following procedure using a case statement classifies the products listed in the Products table of the sample database into one of shirt, hat, shorts, or unknown:

```
CREATE PROCEDURE ProductType (IN product_ID INT, OUT type CHAR(10))
BEGIN
    DECLARE prod_name CHAR(20);
    SELECT Name INTO prod_name FROM GROUP0.Products
    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;
```

The following example uses search conditions to generate a message about product quantity within the SQL Anywhere sample database:

```
CREATE PROCEDURE StockLevel (IN product_ID INT)
BEGIN
    DECLARE qty INT;
    SELECT Quantity INTO qty FROM GROUP0.Products
    WHERE ID = product_ID;
    CASE
    WHEN qty < 30 THEN
        MESSAGE 'Order Stock' TO CLIENT;
    WHEN qty > 100 THEN
        MESSAGE 'Overstocked' TO CLIENT;
    ELSE
        MESSAGE 'Sufficient stock on hand' TO CLIENT;
    END CASE;
END;
```

Related Information

[Unknown values: NULL](#)

A NULL value in a column means that the user or application has made no entry in that column.

[Stored procedures, triggers, batches, and user-defined functions](#)

Procedures and triggers store procedural SQL statements in a database.

[ISNULL function \[Miscellaneous\]](#)

Returns the first non-NULL expression from a list. This function is identical to the COALESCE function.

[BEGIN statement](#)

Specifies a compound statement.

[CASE expressions](#)

The CASE expression provides conditional SQL expressions. Case expressions can be used anywhere an expression can be used.

[CASE statement \[T-SQL\]](#)

Selects an execution path based on multiple cases.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CASE statement [T-SQL]

Selects an execution path based on multiple cases.

Syntax

- **Specify a value expression**

```
CASE value-expression
WHEN [ constant | NULL ] THEN statement-list ...
[ WHEN [ constant | NULL ] THEN statement-list ] ...
[ ELSE statement-list ]
END
```

- **Specify a search condition**

```
CASE
WHEN [ search-condition | NULL ] THEN statement-list ...
[ WHEN [ search-condition | NULL ] THEN statement-list ] ...
[ ELSE statement-list ]
END
```

Remarks

Using a value expression 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. The *value-expression* is an expression that takes on a single value, which may be a string, a number, a date, or other SQL data type. If a WHEN clause exists for the value of *value-expression*, the *statement-list* in the WHEN clause is executed. If no appropriate WHEN clause exists, and an ELSE clause exists, the *statement-list* in the ELSE clause is executed. Execution resumes at the first statement after the END CASE.

If the *value-expression* can be null, use the ISNULL function to replace the NULL *value-expression* with a different expression.

Using a search condition With this form, the statements are executed for the first satisfied *search-condition* in the CASE statement. The ELSE clause is executed if none of the *search-conditions* are met.

If the expression can be NULL, use the following syntax for the first *search-condition*:

```
WHEN search-condition IS NULL THEN statement-list
```

Note Do not confuse the syntax of the CASE statement with that of the CASE expression.

Privileges

None.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** The CASE statement is part of Language Feature P002 (Computational completeness). However, the ANSI/ISO SQL Standard requires END CASE to terminate the CASE statement, rather than END alone.
- **Transact-SQL** Compatible with Adaptive Server Enterprise.

Example

The following procedure using a case statement classifies the products listed in the Products table of the sample database into one of shirt, hat, shorts, or unknown.

```
CREATE PROCEDURE DBA.ProductType( @product_ID INTEGER,@TYPE CHAR(10) OUTPUT ) AS
BEGIN
    DECLARE @prod_name CHAR(20)
    SELECT Name INTO @prod_name FROM GROUP0.Products
    WHERE ID = @product_ID
    IF @prod_name
        = 'Tee Shirt'
        SET @TYPE = 'Shirt'
    ELSE IF @prod_name
        = 'Sweatshirt'
        SET @TYPE = 'Shirt'
    ELSE IF @prod_name
        = 'Baseball Cap'
        SET @TYPE = 'Hat'
    ELSE IF @prod_name
        = 'Visor'
        SET @TYPE = 'Hat'
    ELSE IF @prod_name
        = 'Shorts'
        SET @TYPE = 'Shorts'
    ELSE
        SET @TYPE = 'UNKNOWN'
END;
```

The following example uses a search condition to generate a message about product quantity within the sample database.

```
CREATE PROCEDURE DBA.StockLevel( @product_ID INTEGER ) AS
BEGIN
    DECLARE @qty INTEGER
    SELECT Quantity INTO @qty FROM GROUP0.Products
    WHERE ID = @product_ID
    IF @qty < 30
        MESSAGE 'Order Stock' TO CLIENT
    ELSE IF @qty > 100
        MESSAGE 'Overstocked' TO CLIENT
    ELSE
        MESSAGE 'Sufficient stock on hand' TO CLIENT
END;
```

Related Information

[Unknown values: NULL](#)

A NULL value in a column means that the user or application has made no entry in that column.

[Stored procedures, triggers, batches, and user-defined functions](#)

Procedures and triggers store procedural SQL statements in a database.

[ISNULL function \[Miscellaneous\]](#)

Returns the first non-NULL expression from a list. This function is identical to the COALESCE function.

[BEGIN statement](#)

Specifies a compound statement.

[CASE expressions](#)

The CASE expression provides conditional SQL expressions. Case expressions can be used anywhere an expression can be used.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

CHECKPOINT statement

Checkpoints the database.

Syntax

CHECKPOINT

Remarks

The CHECKPOINT statement forces the database server to execute a checkpoint. Checkpoints are also performed automatically by the database server according to an internal algorithm. It is not normally required for applications to issue the CHECKPOINT statement.

Privileges

You must have the CHECKPOINT system privilege to perform a checkpoint on a database running on a network server (dbsrv).

No privileges are required to perform a checkpoint on a database running on a personal database server (dbeng17).

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.
- **Transact-SQL** The CHECKPOINT statement is supported by Adaptive Server Enterprise.

Related Information

[Database backup and recovery](#)

Use backups to restore all committed changes to a database up to the time it became unavailable.

[Checkpoint logs](#)

The **checkpoint log** is located at the end of the database file and is stored in the system dbspace.

[checkpoint_time option](#)

Sets the maximum number of minutes that the database server runs without doing a checkpoint.

[recovery_time option](#)

Sets the maximum length of time, in minutes, that the database server takes to recover from system failure.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CLEAR statement [Interactive SQL]

Closes any open result sets in Interactive SQL.

Syntax

CLEAR

Remarks

Closes any open result sets and leaves the contents of the SQL Statements pane unchanged

Privileges

None.

Side effects

Closes the cursor associated with the data being cleared.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Related Information

[Interactive SQL](#)

Interactive SQL is a tool that lets you execute SQL statements, run SQL script files, as well as view and compare plans.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CLOSE statement [ESQL] [SP]

Closes a cursor.

Syntax

CLOSE *cursor-name*

cursor-name : *identifier* | *hostvar*

Remarks

This statement closes the named cursor.

The cursor must have been previously opened.

Privileges

None.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Core Feature. When used in Embedded SQL, the CLOSE statement is part of optional Language Feature B031 (Basic dynamic SQL).
- **Transact-SQL** Supported by Adaptive Server Enterprise.

Example

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 CompanyName, CAST(      sum(SalesOrderItems.Quantity *
  Products.UnitPrice) AS INTEGER) VALUE
  FROM GROUP0.Customers
  LEFT OUTER JOIN SalesOrders
  LEFT OUTER JOIN SalesOrderItems
  LEFT OUTER JOIN Products
  GROUP BY CompanyName;
  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
```

Related Information

[OPEN statement \[ESQL\] \[SP\]](#)

Opens a previously declared cursor to access information from the database.

[DECLARE CURSOR statement \[ESQL\] \[SP\]](#)

Declares a cursor.

[PREPARE statement \[ESQL\]](#)

Prepares a statement to be executed later, or defines a cursor.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

COMMENT statement

Stores a comment for a database object in the system tables.

Syntax

```

COMMENT ON {
  COLUMN [ owner.]table-name.column-name
| CERTIFICATE certificate-name
| DBSPACE dbspace-name
| EVENT [ owner.]event-name
| EXTERNAL ENVIRONMENT environment-name
| EXTERNAL [ ENVIRONMENT ] OBJECT object-name
| FOREIGN KEY [ owner.]table-name.key-name
| INDEX [ [ owner. ] table.]index-name
| INTEGRATED LOGIN integrated-login-id
| JAVA CLASS java-class-name
| JAVA JAR java-jar-name
| KERBEROS LOGIN "client-Kerberos-principal"
| LDAP SERVER ldapua-server-name
| LOGIN POLICY policy-name
| MATERIALIZED VIEW [ owner.]materialized-view-name
| MIRROR SERVER mirror-server-name
| ODATA PRODUCER name
| PRIMARY KEY ON [ owner.]table-name
| PROCEDURE [ owner.]procedure-name
| PUBLICATION [ owner.]publication-name
| REMOTE MESSAGE TYPE remote-message-type-name
| ROLE role-name
| SEQUENCE sequence-name
| SERVICE web-service-name
| SPATIAL REFERENCE SYSTEM srs-name
| SPATIAL UNIT OF MEASURE uom-identifier
| SYNCHRONIZATION PROFILE synchronization-profile-name
| TABLE [ owner.]table-name
| TEXT CONFIGURATION [ owner.]text-config-name
| TEXT INDEX text-index-name ON [ owner.]table-name
| TIME ZONE name
| TRIGGER [ [ owner.]tablename.]trigger-name
| USER userid
| VIEW [ owner.]view-name
}
IS comment

comment : string | NULL

```

environment-name :

JAVA

| **PERL**

| **PHP**

| **CLR**

| **C_ESQL32**

| **C_ESQL64**

| **C_ODBC32**

| **C_ODBC64**

Remarks

The COMMENT statement allows you to set a remark (comment) for an object in the database. The COMMENT statement updates remarks listed in the ISYSREMARK system table. You can remove a comment by setting it to NULL. For a comment on an index or trigger, the owner of the comment is the owner of the table on which the index or trigger is defined.

You cannot add comments for local temporary tables.

If you use the **Database Documentation Wizard** to document your database, you have the option to include the comments for procedures, functions, triggers, events, and views in the output.

Privileges

If you have the COMMENT ANY OBJECT system privilege, you can comment on any you can create with the CREATE ANY OBJECT system privilege. If you do not have the COMMENT ANY OBJECT system privilege, you must have the equivalent as noted below:

- For database objects, at least one of the following must be true:
 - you own the object
 - you have the ability to create or alter objects of the same type owned by other users (for example, CREATE ANY TABLE, or ALTER ANY OBJECT)
 - you have the ability to manage objects of that type (for example, MANAGE ANY USER)
- For system roles, you must have the administrative privilege over the role.
- For user-defined roles, you must have the MANAGE ROLES system privilege, or have administrative privilege over the role.
- For Kerberos or integrated logins, you must have the MANAGE ANY USER system privilege.
- For Java classes or jars, you must have the MANAGE ANY EXTERNAL OBJECT system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.
- **Transact-SQL** Not supported by Adaptive Server Enterprise.

Example

The following examples show how to add and remove a comment.

1. Add a comment to the Employees table:

```
COMMENT ON TABLE GROUP0.Employees  
IS 'Employee information';
```

2. Remove the comment from the Employees table:

```
COMMENT  
ON TABLE GROUP0.Employees  
IS NULL;
```

To view the comment set for an object, use a SELECT statement. The following statement retrieves the comment set for the ViewSalesOrders view in the SQL Anywhere sample database.

```
SELECT remarks  
FROM SYSTAB t, SYSREMARK r  
WHERE t.object_id = r.object_id  
AND t.table_name = 'ViewSalesOrders';
```

Related Information

[Documenting a database \(SQL Central\)](#)

Generate an HTML file that shows the dependencies and references for a database by using the Database Documentation Wizard in SQL Central.

[ALTER ODATA PRODUCER statement](#)

Alters an OData Producer.

[DROP ODATA PRODUCER statement](#)

Drops an OData Producer.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

COMMIT statement

Makes changes to the database permanent, or terminates a user-defined transaction.

Syntax

- **Committing work**

COMMIT [WORK]

- **Committing at a transaction level**

COMMIT TRAN[SACTION] [*transaction-name*]

Parameters

transaction-name An optional name assigned to this transaction. It must be a valid identifier. Use transaction names only on the outermost pair of nested BEGIN/COMMIT or BEGIN/ROLLBACK statements.

The following options control the behavior of the COMMIT statement.

- cooperative_commit_timeout option
- cooperative_commits option
- delayed_commits option
- delayed_commit_timeout option

You can use the Commit connection property to return the number of commits on the current connection.

Remarks

- **Committing work** The COMMIT statement ends a transaction and makes all changes made during this transaction permanent in the database.

All data definition statements automatically perform a commit. For information, see the Side effects listing for each SQL statement.

The COMMIT statement fails if the database server detects any invalid foreign keys. This behavior 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 the database option wait_for_commit is set On or a particular foreign key was defined with a CHECK ON COMMIT clause, the database server delays integrity checking until the COMMIT statement is executed.

The use of COMMIT alone is equivalent to COMMIT WORK.

- **Committing at a transaction level** You can use BEGIN TRANSACTION and COMMIT TRANSACTION statements in pairs to construct nested transactions. Nested transactions are similar to savepoints. When executed as the outermost of a set of nested transactions, the statement makes changes to the database permanent. When executed inside a transaction, the COMMIT TRANSACTION statement decreases the nesting level of transactions by one. When transactions are nested, only the outermost COMMIT makes the changes to the database permanent.

Committing at a transaction level is a Transact-SQL extension.

Privileges

None.

Side effects

Closes all cursors except those opened WITH HOLD.

Deletes all rows of declared temporary tables on this connection, unless they were declared using ON COMMIT PRESERVE ROWS.

If the database is not using a transaction log, each COMMIT operation causes an implicit checkpoint.

Standards

- **ANSI/ISO SQL Standard** Committing work is a Core Feature. Committing at a transaction level is a Transact-SQL extension.

Example

The following statement commits the current transaction:

```
COMMIT;
```

The following Transact-SQL batch reports successive values of @@trancount as 0, 1, 2, 1, 0.

```
PRINT @@trancount
BEGIN TRANSACTION
PRINT @@trancount
BEGIN TRANSACTION
PRINT @@trancount
COMMIT TRANSACTION
PRINT @@trancount
COMMIT TRANSACTION
PRINT @@trancount
go
```

Related Information

[Permanent data changes](#)

Use the COMMIT statement after groups of statements that make sense together. The COMMIT statement makes database changes permanent.

[BEGIN TRANSACTION statement \[T-SQL\]](#)

Begins a user-defined transaction.

[wait_for_commit option](#)

Determines when foreign key integrity is checked, as data is manipulated.

[auto_commit option \[Interactive SQL\]](#)

Controls whether a COMMIT is performed after each statement.

[commit_on_exit option \[Interactive SQL\]](#)

Controls the behavior when Interactive SQL disconnects or shuts down.

[Keyboard shortcuts \(Interactive SQL\)](#)

Interactive SQL provides several keyboard shortcuts.

[SAVEPOINT statement](#)

Establishes a savepoint within the current transaction.

[List of connection properties](#)

Connection properties are available for each connection to a database. Connection property names are case insensitive.

[PREPARE TO COMMIT statement](#)

Checks whether a COMMIT can be performed successfully.

[ROLLBACK statement](#)

Ends a transaction and undo any changes made since the last COMMIT or ROLLBACK.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CONFIGURE statement [Interactive SQL]

Opens the Interactive SQL **Options** window.

Syntax

CONFIGURE

Remarks

The CONFIGURE statement opens the Interactive SQL **Options** window. This window displays the current settings of all Interactive SQL options. It does not display or allow you to modify database options. You can configure Interactive SQL settings in this window.

Privileges

None.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Related Information

[Interactive SQL](#)

Interactive SQL is a tool that lets you execute SQL statements, run SQL script files, as well as view and compare plans.

[Customizing Interactive SQL](#)

Configure how Interactive SQL result sets are displayed, disable warning messages, and set Interactive SQL as the default editor for *.sql* files.

[SET OPTION statement](#)

Changes the values of database and connection options.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CONNECT statement [ESQL] [Interactive SQL]

Establishes a connection to a database.

Syntax

- **Shared memory connections**

CONNECT

[**TO** *database-server-name*]

[**DATABASE** *database-file*]

[**AS** *connection-name*]

[**USER**] *userid* [**IDENTIFIED BY** *password*]

database-server-name, *database-file*, *connection-name*, *userid*, *password* :
{ *identifier* | *string* | *hostvar* }

- **TCP/IP connections**

CONNECT USING *connect-string*

connect-string : { *identifier* | *string* | *hostvar* }

Parameters

AS clause 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.

For TCP/IP connections, a *connect-string* is a list of parameter settings of the form keyword=value, separated by semicolons, and must be enclosed in single quotes.

Remarks

The CONNECT statement establishes a connection to the database identified by *database-file* running on the database server identified by *database-server-name*. This statement is not supported in procedures, triggers, events, or batches.

Shared memory connections are only supported for connections to database servers running on the same computer. To connect to a local database server using TCP/IP or to a database server running on a different computer, use the syntax for TCP/IP connections.

- **Embedded SQL behavior** In Embedded SQL, if no *database-server-name* is specified, the default local database server is assumed (the first database server started). If no *database-file* is specified, the first database on the given server is assumed.

The WHENEVER statement, SET SQLCA, and some DECLARE statements do not generate code and may appear before the CONNECT statement in the source file. Otherwise, no statements are allowed until a successful CONNECT statement has been executed.

The user ID and password are used for privilege checks on all dynamic SQL statements.

Note For SQL Anywhere, only shared memory connections are supported with

Embedded SQL. For UltraLite, both shared memory and TCP/IP connections can be used with Embedded SQL.

- **Interactive SQL behavior** If no database or server is specified in the CONNECT statement, Interactive SQL remains connected to the current database, rather than to the default server and database. If a database name is specified without a server name, Interactive SQL attempts to connect to the specified database on the current server. If a server name is specified without a database name, Interactive SQL connects to the default database on the specified server.

For example, if the following batch is executed while connected to a database, the two tables are created in the same database.

```
CREATE TABLE t1( c1 int );  
CONNECT DBA IDENTIFIED BY passwd;  
CREATE TABLE t2 (c1 int );
```

No other database statements are allowed until a successful CONNECT statement has been executed.

When Interactive SQL is run in windowed mode, you are prompted for any missing connection parameters.

When Interactive SQL is running in command-prompt mode (-nogui is specified when you start Interactive SQL from a command line) or batch mode, or if you execute CONNECT without an AS clause, an unnamed connection is opened. If there is another unnamed connection already opened, the old one is automatically closed. Otherwise, existing connections are not closed when you execute a CONNECT statement.

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.

When connecting to Interactive SQL, specifying CONNECT [USER] *userid* is the same as executing a SETUSER WITH OPTION *userid* statement.

In Interactive SQL, the connection information (including the database name, your user ID, and the database server) appears in the title bar above the SQL Statements pane. If you are not connected to a database, Not Connected appears in the title bar.

Note Both syntaxes are valid with Interactive SQL except that Interactive SQL does not support the *hostvar* argument.

This SQL statement is not supported for SAP HANA databases.

If you use this statement in a procedure, do not specify the password (IDENTIFIED BY clause) as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

Privileges

None.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Shared memory connections is an optional ANSI/ISO SQL Language Feature F771. TCP/IP connections is not in the standard.
- **Transact-SQL** Both syntaxes are supported by Adaptive Server Enterprise.

Example

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 "passwd";
```

The following examples assume that the SQL Anywhere sample database has already been started. Connect to a database from Interactive SQL. Interactive SQL prompts for a user ID and a password.

```
CONNECT;
```

Connect to the default database as user DBA from Interactive SQL. Interactive SQL prompts for a password.

```
CONNECT USER "DBA";
```

Connect to the sample database as user DBA from Interactive SQL.

```
CONNECT  
TO demo17  
USER DBA  
IDENTIFIED BY sql;
```

Connect to the sample database using a connection string, from Interactive SQL.

```
CONNECT  
USING 'UID=DBA;PWD=sql;DBN=demo';
```

Related Information

[Connection parameters](#)

Connection parameters are included in connection strings.

[Interactive SQL](#)

Interactive SQL is a tool that lets you execute SQL statements, run SQL script files, as well as view and compare plans.

[GRANT CONNECT statement](#)

Creates a new user, and can also be used by a user to change their own password. However, it is recommended that you use the CREATE USER statement to create users instead of the GRANT CONNECT statement.

[DISCONNECT statement \[ESQL\] \[Interactive SQL\]](#)

Drops a connection to a database.

[SET CONNECTION statement](#) [\[Interactive SQL\]](#) [\[ESQL\]](#)

Changes the active database connection.

[SETUSER statement](#)

Allows a user to assume the identity of (impersonate) another authorized user.

[Troubleshooting: connections](#)

An understanding of how connections are established can help you resolve connectivity problems.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

CONTINUE statement

Restarts a loop.

Syntax

CONTINUE [*statement-label*]

Remarks

The CONTINUE statement is a control statement that restarts a loop. Execution continues at the first statement in the loop.

When CONTINUE appears within a set of statements using Transact-SQL, do not use *statement-label*.

Privileges

None.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.
- **Transact-SQL** CONTINUE without a statement label is supported by SAP Adaptive Server Enterprise.

Example

The following fragment shows how the CONTINUE statement restarts a loop. This example displays the odd numbers between 1 and 10.

```
BEGIN
  DECLARE i INT;
  SET i = 0;
  lbl:
  WHILE i < 10 LOOP
    SET i = i + 1;
    IF mod( i, 2 ) = 0 THEN
      CONTINUE lbl
    END IF;
    MESSAGE 'The value ' || i || ' is odd.' TO CLIENT;
  END LOOP lbl;
END
```

Related Information

[Stored procedures, triggers, batches, and user-defined functions](#)

Procedures and triggers store procedural SQL statements in a database.

[LOOP statement](#)

Repeats the execution of a statement list.

[WHILE statement \[T-SQL\]](#)

Provides repeated execution of a statement or compound statement.

[FOR statement](#)

Repeats the execution of a statement list once for each row in a cursor.

[BEGIN statement](#)

Specifies a compound statement.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CREATE CERTIFICATE statement

Adds or replaces a certificate in the database from the given file or string. To create a certificate, use the Certificate Creation utility (createcert).

Syntax

```
CREATE [ OR REPLACE ] CERTIFICATE certificate-name  
FROM { certificate-string | variable-name | FILE file-name }
```

Parameters

FROM clause This clause specifies a file, string, or variable containing a certificate.

Remarks

The CREATE CERTIFICATE statement adds or replaces a certificate in the database from the given file, string, or variable. The file, string, or variable should contain either a binary DER-format certificate or a text PEM-format certificate. DER-format certificates are converted and stored as PEM certificates.

Certificates that are stored in the database can be used by web service procedures and functions that make secure HTTPS connections to a web server. They can also be used to send secure messages using the xp_startsmtp system procedure.

When you add a certificate, it is added to the ISYSCERTIFICATE system table. Use the corresponding system view SYSCERTIFICATE to view the table.

The CREATE CERTIFICATE statement is not used to create an actual certificate. Use the Certificate Creation utility (createcert) to do this.

Privileges

You must have the MANAGE CERTIFICATES system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example creates a certificate called mycert in the database using the contents of the specified certificate file.

```
CREATE CERTIFICATE mycert  
FROM FILE 'C:\\Users\\Public\\Documents\\SQL Anywhere  
17\\Samples\\Certificates\\rsaroot.crt';
```

Related Information

[DROP CERTIFICATE statement](#)

Drops a certificate from the database.

[CREATE PROCEDURE statement \[Web service\]](#)

Creates a user-defined web client procedure that makes HTTP or SOAP requests to an HTTP server.

[CREATE FUNCTION statement \[Web service\]](#)

Creates a web client function that makes an HTTP or SOAP over HTTP request.

[xp_startsmtp system procedure](#)

Starts an email session under SMTP.

[SYSCERTIFICATE system view](#)

Each row of the SYSCERTIFICATE system view stores a certificate in text PEM-format. The underlying system table for this view is ISYSCERTIFICATE.

[sa_certificate_info system procedure](#)

Displays information about the specified certificate that is stored in the database.

[Certificate Creation utility \(createcert\)](#)

Creates X.509 certificates.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

CREATE DATABASE statement

Creates a database.

Syntax

```
CREATE DATABASE db-filename-string  

DBA USER userid  

DBA PASSWORD password  

[ create-option ... ]
```

create-option :

```
[ ACCENT { RESPECT | IGNORE | FRENCH } ]  

[ ASE [ COMPATIBLE ] ]  

[ BLANK PADDING { ON | OFF } ]  

[ CASE { RESPECT | IGNORE } ]  

[ CHECKSUM { ON | OFF } ]  

[ COLLATION collation-label [ ( collation-tailoring-string ) ] ]  

[ DATABASE SIZE size { KB | MB | GB | PAGES | BYTES } ]  

[ ENCODING encoding-label ]  

[ ENCRYPTED [ TABLE ] { algorithm-key-spec | OFF } ]  

[ JCONNECT { ON | OFF } ]  

[ MINIMUM PASSWORD LENGTH positive-integer ]  

[ MIRROR mirror-filename-string ]  

[ PAGE SIZE page-size ]  

[ NCHAR COLLATION nchar-collation-label [ ( collation-tailoring-string ) ] ]  

[ SYSTEM PROCEDURE AS DEFINER { ON | OFF } ]  

[ [ TRANSACTION ] { LOG OFF | LOG ON [ log-filename-string ] ]
```

page-size :

2048 | **4096** | **8192** | **16384** | **32768**

algorithm-key-spec :

```
ON  

| [ ON ] KEY key [ ALGORITHM AES-algorithm ]  

| [ ON ] ALGORITHM AES-algorithm KEY key  

| [ ON ] ALGORITHM 'SIMPLE'
```

AES-algorithm :

'AES' | **'AES256'** | **'AES_FIPS'** | **'AES256_FIPS'**

Parameters

CREATE DATABASE The file names (*db-filename-string*, *log-filename-string*, and *mirror-filename-string*) are strings containing operating system file names. As literal strings, they must be enclosed in single quotes.

- If you specify a path, any backslash characters (\) must be doubled if they are followed by an n or an x. Escaping them prevents them from being interpreted as new line characters (\n) or as hexadecimal numbers (\x), according to the rules for strings in SQL.

Here are some examples where this is important.

```
CREATE DATABASE 'c:\\temp\\x41\\x42\\x43xyz.db'  
DBA USER 'DBA' DBA PASSWORD 'passwd';
```

The initial \\ sequence represents a backslash. The \\x sequences represent the characters A, B, and C, respectively. The file name here is *ABCxyz.db*.

```
CREATE DATABASE 'c:\\temp\\nest.db'  
DBA USER 'DBA' DBA PASSWORD 'passwd';
```

To avoid having the \\n sequence interpreted as a newline character, the backslash is doubled.

It is always safer to escape the backslash character. For example:

```
CREATE DATABASE 'c:\\my_db.db'  
DBA USER 'DBA' DBA PASSWORD 'passwd'  
LOG ON 'e:\\logdrive\\my_db.log';
```

- If you do not specify a path, or a relative path, the database file is created relative to the working directory of the database server. If you specify no path for a transaction log file, the file is created in the same directory as the database file. Store the database files and the transaction log on separate disks on the computer.
- If you provide no file extension, a file is created with extension *.db* for databases, *.log* for the transaction log, and *.mlg* for the transaction log mirror.
- The directory path is relative to the database server.

You cannot specify *utility_db* for *db-filename-string*. This name is reserved for the utility database.

ACCENT clause This clause is used to specify accent sensitivity for the database. Support for this clause is deprecated. Use the collation tailoring options provided for the COLLATION and NCHAR COLLATION clauses to specify accent sensitivity.

The ACCENT clause applies only when using the UCA (Unicode Collation Algorithm) for the collation specified in the COLLATION or NCHAR COLLATION clause. ACCENT RESPECT causes the UCA string comparison to respect accent differences between letters. For example, e is less than é. ACCENT FRENCH is similar to ACCENT RESPECT, except that accents are compared from right to left, consistent with the rules of the French language. ACCENT IGNORE causes string comparisons to ignore accents. For example, e is equal to é.

If accent sensitivity is not specified when the database is created, the default accent sensitivity for comparisons and sorting is *insensitive*, with one exception; for Japanese databases created with a UCA collation, the default accent sensitivity is *sensitive*.

ASE COMPATIBLE clause Do not create the SYS.SYSCOLUMNS and SYS.SYSINDEXES views. By default, these views are created for compatibility with system tables available in Watcom SQL (version 4 and earlier of this software). These views conflict with the Adaptive Server Enterprise compatibility views *dbo.syscolumns* and *dbo.sysindexes*.

BLANK PADDING clause The database server compares all strings as if they are varying length and stored using the VARCHAR domain. This includes string comparisons involving fixed length CHAR or NCHAR columns. In addition, the database server never trims or pads values with

trailing blanks when the values are stored in the database.

By default, the database server treats blanks as significant characters. For example, the value 'a ' (the character 'a' followed by a blank) is not equivalent to the single-character string 'a'. Inequality comparisons also treat a blank as any other character in the collation.

If blank padding is enabled (specifying `BLANK PADDING ON`), the semantics of string comparisons more closely follow the ANSI/ISO SQL standard. With blank-padding enabled, the database server ignores trailing blanks in any comparison.

In the example above, an equality comparison of 'a ' to 'a' in a blank-padded database returns `TRUE`. With a blank-padded database, fixed-length string values are padded with blanks when they are fetched by an application. Whether the application receives a string truncation warning on such an assignment is controlled by the `ansi_blanks` connection option.

CASE clause This clause is used to specify case sensitivity for the database. Support for this clause is deprecated. Use the collation tailoring options provided for the `COLLATION` and `NCHAR COLLATION` clauses to specify case sensitivity.

`CASE RESPECT` causes case-sensitive string comparisons for all `CHAR` and `NCHAR` data types. Comparisons using `UCA` consider the case of a letter only if the base letters and accents are all equal. For all other collations, uppercase and lowercase letters are distinct; for example, a is less than A, which is less than b, and so on. `CASE IGNORE` causes case-insensitive string comparisons. Uppercase and lowercase letters are considered to be exactly equal.

If case sensitivity is not specified when the database is created, default case sensitivity for comparisons and sorting is *insensitive*, with one exception; for Japanese databases created with a `UCA` collation, default case sensitivity is *sensitive*.

`CASE RESPECT` is provided for compatibility with the ISO/ANSI SQL standard. Identifiers in the database are always case insensitive, even in case-sensitive databases.

CHECKSUM clause Checksums are used to determine whether a database page has been modified on disk. When you create a database with global checksums enabled, a checksum is calculated for each page just before it is written to disk. The next time the page is read from disk, the page's checksum is recalculated and compared to the checksum stored on the page. If the checksums are different, then the page has been modified on disk and an error occurs. Databases created with global checksums enabled can also be validated using checksums. You can check whether a database was created with global checksums enabled by executing the following statement:

```
SELECT DB_PROPERTY ( 'Checksum' );
```

This query returns `ON` if global checksums are turned on, otherwise, it returns `OFF`. Global checksums are turned on by default, so if the `CHECKSUM` clause is omitted, `ON` is applied.

Regardless of the setting of this clause, the database server always enables write checksums for databases running on storage devices such as removable drives, to help provide early detection if the database file becomes corrupt. The database server also calculates checksums for critical pages during validation activities.

For databases that do not have global checksums enabled, you can enable write checksums by using the `-wc` options.

COLLATION clause The collation specified by the `COLLATION` clause is used for sorting and comparison of character data types (`CHAR`, `VARCHAR`, and `LONG VARCHAR`). The collation provides character comparison and ordering information for the encoding (character set) being

used. If the COLLATION clause is not specified, the database server chooses a collation based on the operating system language and encoding.

The collation can be chosen from the list of collations that use the SQL Anywhere Collation Algorithm (SACA), or it can be the Unicode Collation Algorithm (UCA). If UCA is specified, also specify the ENCODING clause.

It is important to choose your collation carefully. It cannot be changed after the database has been created.

Optionally, you can specify collation tailoring options (*collation-tailoring-string*) for additional control over the sorting and comparing of characters. These options take the form of keyword=value pairs, assembled in parentheses, following the collation name. For example, ...
`CHAR COLLATION 'UCA(locale=es;case=respect;accent=respect)'.`

DATABASE SIZE clause Use this optional clause to set the initial size of the database file. You can use KB, MB, GB, or PAGES to specify units of kilobytes, megabytes, gigabytes, or pages respectively.

Specifying the file size at creation time is a way of preallocating space for the file. This helps reduce the risk of running out of space on the drive the database is located on. As well, it can help improve performance by increasing the amount of data that can be stored in the database before the database server needs to grow the database, which can be a time-consuming operation.

DBA USER and DBA PASSWORD clauses Use these clauses to specify a DBA user ID and password for the database.

By default, passwords must be a minimum length of 6 characters unless the MINIMUM PASSWORD LENGTH clause is specified and set to a different value. Passwords should be composed of 7-bit ASCII characters. Other characters may not work correctly if the server cannot convert from the client character set to UTF-8.

ENCODING clause Most collations specified in the COLLATION clause dictate both the encoding (character set) and ordering. For those collations, the ENCODING clause should not be specified. However, if the value specified in the COLLATION clause is UCA (Unicode Collation Algorithm), use the ENCODING clause to specify a locale-specific encoding and get the benefits of the UCA for comparison and ordering. The ENCODING clause may specify UTF-8 or any single-byte encoding for CHAR data types. ENCODING may not specify a multibyte encoding other than UTF-8.

If you choose the UCA collation, you can optionally specify collation tailoring options.

If COLLATION is set to UCA and ENCODING is not specified, then the database server uses UTF-8.

ENCRYPTED or ENCRYPTED TABLE clause Encryption makes stored data undecipherable. Use the ENCRYPTED keyword (without TABLE) when you want to encrypt the entire database. Use the ENCRYPTED TABLE clause when you only want to enable table encryption. Enabling table encryption means that the tables that are subsequently created or altered using the ENCRYPTED clause are encrypted using the settings you specified at database creation.

There are two levels of database and table encoding: simple obfuscation and strong encryption. Obfuscation is not encryption, and someone with cryptographic expertise could decipher the data. Strong encryption ensures that the data is unreadable and virtually undecipherable.

For simple obfuscation, specify ENCRYPTED ON ALGORITHM SIMPLE, or ENCRYPTED ALGORITHM SIMPLE, or specify the ENCRYPTED ON clause without specifying an algorithm or key.

For strong encryption, specify **ENCRYPTED ON ALGORITHM** with a 128-bit or 256-bit AES algorithm, and the **KEY** clause to specify an encryption key. Choose a value for your key that is at least 16 characters long, contains a mix of uppercase and lowercase, and includes numbers, letters, and special characters. A key can be specified as either a string or a variable name.

Caution

For strongly encrypted databases, be sure to store a copy of the key in a safe location. If you lose the encryption key there is no way to access the data, even with the assistance of Technical Support. The database must be discarded and you must create a new database.

You can also create an encrypted copy of an existing database using the **CREATE ENCRYPTED DATABASE** statement.

JCONNECT clause To allow the jConnect JDBC driver access to system catalog information, specify **JCONNECT ON**. This clause installs the system objects that provide jConnect support. Specify **JCONNECT OFF** to exclude the jConnect system objects. You can still use JDBC, as long as you do not access system information. **JCONNECT** is **ON** by default.

MINIMUM PASSWORD LENGTH clause Use this clause to set the minimum password length. If this clause is not specified, then the default minimum password length for a new database is 6.

PAGE SIZE clause The page size for a database can be 2048, 4096, 8192, 16384, or 32768 bytes. The default page size is 4096 bytes. The 2048 page size is deprecated. Large databases generally obtain performance benefits from a larger page size, but there can be additional overhead associated with the large page sizes.

For example:

```
CREATE DATABASE 'c:\\temp\\my_db.db'  
DBA USER 'DBA' DBA PASSWORD 'passwd'  
PAGE SIZE 4096;
```

Note The page size cannot be larger than the page size used by the current server. The server page size is taken from the first set of databases started, or is set on the server command line using the **-gp** option.

NCHAR COLLATION clause The collation specified by the **NCHAR COLLATION** clause is used for sorting and comparing national character data types (**NCHAR**, **NVARCHAR**, and **LONG NVARCHAR**). The collation provides character ordering information for the UTF-8 encoding (character set) used for national characters. If the **NCHAR COLLATION** clause is not specified, the database server uses the Unicode Collation Algorithm (UCA). The only other allowed collation is **UTF8BIN**, which provides a binary ordering of all characters whose encoding is greater than 0x7E.

Optionally, you can specify collation tailoring options (*collation-tailoring-string*) for additional control over the sorting and comparing of characters. These options take the form of keyword=value pairs, assembled in a quoted string following the collation name. For example, `... NCHAR COLLATION 'UCA(locale=es;case=respect;accent=respect)'`. If you specify the **ACCENT** or **CASE** clause and a collation tailoring string that contains settings for case and accent, the values of the **ACCENT** and **CASE** clauses are used as defaults only.

Note

When you specify the UCA collation, all collation tailoring options are supported. For all other collations, only the case sensitivity tailoring option is supported.

Databases created with collation tailoring options cannot be started using a pre-10.0.1 database server.

SYSTEM PROCEDURE AS DEFINER { ON | OFF } clause The SYSTEM PROCEDURE AS DEFINER clause specifies whether to execute pre-16.0 system procedures that perform privileged tasks with the privileges of the invoker or the definer (owner). ON means that these system procedures are executed with the privileges of the definer (owner). OFF means these system procedures are executed with the privileges of the invoker.

If this clause is not specified, the default is to run these procedures with the privileges of the invoker.

This setting does not impact user-defined procedures, or any system procedures introduced in version 16.0 or later.

TRANSACTION LOG clause The transaction log is a file where the database server logs all changes made to the database. The transaction log plays a key role in backup and recovery, and in data replication.

The MIRROR clause of the TRANSACTION clause allows you to provide a file name if you are using 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, the database server does not use a transaction log mirror.

Remarks

Creates a database file with the supplied name and attributes. The database is stored as an operating system file. This statement is not supported in procedures, triggers, events, or batches.

You must be connected to a database to create another database. For example, connect to the utility database.

The account under which the database server is running must have write permissions on the directories where files are created.

Messages sent to the client indicate what type of database encryption is used for the database. If encryption is used, the algorithm being used is also displayed.

Privileges

Your ability to execute this statement depends on the setting for the -gu database option, and whether you have the SERVER OPERATOR system privilege.

Side effects

An operating system file is created.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.
- **Transact-SQL** The CREATE DATABASE statement is supported by Adaptive Server Enterprise, though with different clauses.

Example

The following statement creates a database file named *temp.db* in the *C:\temp* directory:

```
CREATE DATABASE 'c:\\temp\\temp.db'  
DBA USER 'DBA' DBA PASSWORD 'passwd';
```

The following statement creates a database file named *mydb.db* in the *C:\temp* directory.

```
CREATE DATABASE 'C:\\temp\\mydb.db'  
DBA USER 'DBA' DBA PASSWORD 'passwd'  
TRANSACTION LOG ON  
CASE IGNORE  
PAGE SIZE 4096  
ENCRYPTED OFF  
BLANK PADDING OFF;
```

The following statement creates a database using code page 1252 and uses the UCA for both CHAR and NCHAR data types. Accents and case are respected during comparison and sorting.

```
CREATE DATABASE 'c:\\temp\\uca.db'  
DBA USER 'DBA' DBA PASSWORD 'passwd'  
COLLATION 'UCA'  
ENCODING 'CP1252'  
NCHAR COLLATION 'UCA'  
ACCENT RESPECT  
CASE RESPECT;
```

The following statement creates a database, *myencrypteddb.db*, that is encrypted using simple obfuscation:

```
CREATE DATABASE 'c:\\temp\\myencrypteddb.db'  
DBA USER 'DBA' DBA PASSWORD 'passwd'  
ENCRYPTED ON;
```

The following statement creates a database, *mystrongencryptdb.db*, that is encrypted using the key gh67AB2 (strong encryption):

```
CREATE DATABASE 'c:\\temp\\mystrongencryptdb.db'  
DBA USER 'DBA' DBA PASSWORD 'passwd'  
ENCRYPTED ON KEY 'gh67AB2';
```

The following statement creates a database, *mytableencryptdb.db*, with table encryption enabled using simple obfuscation. Notice the keyword TABLE inserted after ENCRYPTED to indicate table encryption instead of database encryption:

```
CREATE DATABASE 'c:\\temp\\mytableencryptdb.db'  
DBA USER 'DBA' DBA PASSWORD 'passwd'  
ENCRYPTED TABLE ON;
```

The following statement creates a database, *mystrongencrypttabledb.db*, with table encryption enabled using simple encryption:

```
CREATE DATABASE 'c:\\temp\\mystrongencrypttabledb.db'  
DBA USER 'DBA' DBA PASSWORD 'passwd'  
ENCRYPTED TABLE ON 'SIMPLE';
```

The following statement creates a database file named *mydb.db* that uses collation 1252LATIN1. The NCHAR collation is set to UCA, with the locale set to es, and has case sensitivity and accent sensitivity enabled:

```
CREATE DATABASE 'c:\\temp\\my2.db'  
DBA USER 'DBA' DBA PASSWORD 'passwd'  
COLLATION '1252LATIN1(case=respect)'  
NCHAR COLLATION 'UCA(locale=es;case=respect;accent=respect)';
```

The following statement creates a database with a Greek collation:

```
CREATE DATABASE 'c:\\temp\\mydb.db'  
DBA USER 'DBA' DBA PASSWORD 'passwd'  
COLLATION '1253ELL';
```

The following statement creates a database named *mydb.db* with a transaction log mirror:

```
CREATE DATABASE 'c:\\mydb.db'  
DBA USER 'DBA' DBA PASSWORD 'passwd'  
TRANSACTION LOG ON 'mydb.log'  
MIRROR 'd:\\mydb.mlg';
```

Related Information

[Running pre-16.0 system procedures as invoker or definer](#)

Some system procedures present in the software before version 16.0 that perform privileged tasks in the database, such as altering tables, can be run with either the privileges of the invoker, or of the definer (owner).

[Escape sequences](#)

Sometimes you must put characters into string literals that cannot be typed or entered normally. Examples include control characters (such as a new line character), single quotes (which would otherwise mark the end of the string literal), and hexadecimal byte values. For this purpose, you use an escape sequence.

[Corruption detection using checksums](#)

Checksums are used to determine whether a database page has been modified on disk.

[Table encryption](#)

Table encryption allows you to encrypt tables or materialized views with sensitive data, without the performance impact that encrypting the entire database might cause.

[The utility database \(utility_db\)](#)

The **utility database** is a phantom database with no physical representation.

[Collation considerations](#)

You specify the collation for a database when you create the database.

[Database encoding options](#)

Two methods of database encoding are supported: simple obfuscation and strong encryption.

[International languages and character sets](#)

Internationalization refers to the ability of software to handle a variety of languages and their appropriate characters sets, independently of the language in which the software is running, or the operating system on which the software is running.

[ALTER DATABASE statement](#)

Upgrades the database, turns jConnect support for a database on or off, calibrates the database, changes the transaction log and transaction log mirror file names, or forces a mirror server to take ownership of a database.

[CREATE ENCRYPTED DATABASE statement](#)

Creates an encrypted copy of an existing database, including all transaction logs and dbspaces; or creates a copy of an existing database with table encryption enabled.

[Collation tailoring options](#)

If you choose the UCA collation when you create a database, you can optionally specify collation tailoring options.

[Recommended character sets and collations](#)

While the names of hundreds of character sets, code pages, encodings, and collations are recognized, there are several that are recommended for use with Windows and Unix platforms, depending on the language in use.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CREATE DBSPACE statement

Defines a new database space and creates the associated database file.

Syntax

CREATE DBSPACE *dbspace-name* **AS** *filename*

Parameters

dbspace-name Specify a name for the dbspace. This is not the actual database file name, which you specify using *filename*. *dbspace-name* is an internal name you can refer to, for example in statements and procedures. You cannot use the following names for a dbspace because they are reserved for predefined dbspaces: system, temporary, temp, translog, and translogmirror.

An error is returned if you specify a value that contains a period (.).

filename Specify a name for the database file, including, optionally, the path to the file. If no path is specified, the database file is created in the same location (directory) as the main database file. If you specify a different location, the path is relative to the database server. The backslash (\) is an escape character in SQL strings, so each backslash must be doubled.

The *filename* parameter must be either a string literal or a variable.

Remarks

The CREATE DBSPACE statement creates a new database file. When a database is created, 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 main file, which means that the database can be larger than one physical device.

If disk sandboxing is enabled, then database operations are limited to the directory where the main database file is located.

For each database, there is a limit of twelve dbspaces in addition to the main file.

Each object, such as a table or index, is contained entirely within one dbspace. The IN clause of the CREATE statement specifies the dbspace into which an object is placed. Objects are put into the system database file by default. You can also specify which dbspace tables are created in by setting the default_dbspace option before you create the tables.

Privileges

You must have the MANAGE ANY DBSPACE system privilege.

Side effects

Automatic commit. Automatic checkpoint.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example creates a dbspace called libbooks in the c:\ directory. A subsequent CREATE

TABLE statement creates a table called LibraryBooks in the libbooks dbspace.

```
CREATE DBSPACE libbooks
AS 'c:\\library.db';
CREATE TABLE LibraryBooks (
    title char(100),
    author char(50),
    isbn char(30),
) IN libbooks;
```

Related Information

[Strings](#)

A string is a sequence of characters up to 2 GB in size.

[Predefined dbspaces](#)

The database server uses predefined dbspaces for its databases.

[Additional dbspaces considerations](#)

Additional database files allow you to cluster related information in separate files.

[DROP DBSPACE statement](#)

Removes a dbspace from the database.

[default_dbspace option](#)

Changes the default dbspace in which tables are created.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CREATE DECRYPTED DATABASE statement

Creates a decrypted copy of an existing database, including all transaction logs and dbspaces.

Syntax

```
CREATE DECRYPTED DATABASE newfile  
FROM oldfile  
[ KEY key ]
```

Parameters

FROM clause Use this clause to specify the name of the database to copy (*oldfile*).

KEY clause Use this clause to specify the encryption key needed to decrypt the database. You can specify either a string or a variable name for the key. You do not specify the KEY clause if the existing database was encoded with simple obfuscation, which does not require a key.

Remarks

The CREATE DECRYPTED DATABASE statement produces a new database file (*newfile*), and does not replace or remove the original database file (*oldfile*).

All encrypted tables in *oldfile* are not encrypted in *newfile*, and table encryption is not enabled.

Note For databases created with SQL Anywhere 12 or later, the ISYSCOLSTAT, ISYSUSER, and ISYSEXTERNLOGIN system tables always remain encrypted to protect the data from unauthorized access.

If *oldfile* uses a transaction log or transaction log mirror, the files are renamed *newfile.log* and *newfile.mlg*, respectively.

If *oldfile* contains dbspace files, a D (decrypted) is added to the file name. For example, when you execute the CREATE DECRYPTED DATABASE statement, the file *mydbspace.dbs* is changed to *mydbspace.dbsD*.

If disk sandboxing is enabled, then the database's operations are limited to the directory where the main database file is located.

You cannot execute this statement on a database that requires recovery. This statement is not supported in procedures, triggers, events, or batches.

You cannot be connected to the database you are decrypting. You must be connected to a different database. For example, connect to the utility database. The database that you are encrypting must not be running.

Privileges

Your ability to execute this statement depends on the setting for the -gu database option, and whether you have the SERVER OPERATOR system privilege.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The first statement below creates an AES256-encrypted copy of the *demo.db* called *demoEncrypted.db*. The second statement creates a decrypted copy of *demoEncrypted.db* called *demoDecrypted.db*.

```
CREATE ENCRYPTED DATABASE 'demoEncrypted.db'  
  FROM 'demo.db'  
  KEY 'Sd8f6654*Mnn'  
  ALGORITHM 'AES256';  
CREATE DECRYPTED DATABASE 'demoDecrypted.db'  
  FROM 'demoEncrypted.db'  
  KEY 'Sd8f6654*Mnn';
```

Related Information

[Database encryption and decryption](#)

You can encrypt your database to make it more difficult for someone to decipher the data in your database.

[CREATE ENCRYPTED DATABASE statement](#)

Creates an encrypted copy of an existing database, including all transaction logs and dbspaces; or creates a copy of an existing database with table encryption enabled.

[CREATE ENCRYPTED FILE statement](#)

Creates a strongly encrypted copy of a database file, transaction log, transaction log mirror, or dbspace.

[CREATE DECRYPTED FILE statement](#)

Creates a decrypted copy of a strongly encrypted database, and can be used to create decrypted copies of transaction logs, transaction log mirrors, and dbspaces.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CREATE DECRYPTED FILE statement

Creates a decrypted copy of a strongly encrypted database, and can be used to create decrypted copies of transaction logs, transaction log mirrors, and dbspaces.

Syntax

```
CREATE DECRYPTED FILE newfile  
FROM oldfile KEY key
```

Parameters

FROM clause Lists the file name of the encrypted file.

KEY clause Lists the key required to access the encrypted file. The key can be either a string or a variable name.

Remarks

Use this statement when your database requires recovery and you need to create a decrypted copy of the database for support reasons. You must also use this statement to decrypt any database-related files such as the transaction log, transaction log mirror, or dbspaces.

The original database file must be strongly encrypted using an encryption key. The resulting file is an exact copy of the encrypted file, without encryption and therefore requiring no encryption key.

If disk sandboxing is enabled, then database operations are limited to the directory where the main database file is located.

If a database is decrypted using this statement, the corresponding transaction log file (and any dbspaces) must also be decrypted to use the database.

If a database requiring recovery is decrypted, its transaction log file must also be decrypted and recovery on the new database is necessary. The name of the transaction log file remains the same in this process, so if the database and transaction log file are renamed, then you need to run `dblog -t` on the resulting database.

You cannot use this statement on a database that has table encryption enabled. If you have tables you want to decrypt, use the NOT ENCRYPTED clause of the ALTER TABLE statements to decrypt them.

Note For databases created with SQL Anywhere 12 or later, the ISYSCOLSTAT, ISYSUSER, and ISYSEXTERNLOGIN system tables always remain encrypted to protect the data from unauthorized access to the database file.

This statement is not supported in procedures, triggers, events, or batches.

You cannot be connected to the database you are decrypting. You must be connected to a different database. For example, connect to the utility database. The database that you are encrypting must not be running.

Privileges

Your ability to execute this statement depends on the setting for the `-gu` database option, and whether you have the SERVER OPERATOR system privilege.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example decrypts a fictitious encrypted database called encContacts, and creates a new unencrypted database called contacts.

```
CREATE DECRYPTED FILE 'contacts.db'  
FROM 'encContacts.db'  
KEY 'Sd8f6654*Mnn';
```

Related Information

[ALTER TABLE statement](#)

Modifies a table definition or disables dependent views.

[CREATE ENCRYPTED FILE statement](#)

Creates a strongly encrypted copy of a database file, transaction log, transaction log mirror, or dbspace.

[CREATE DECRYPTED DATABASE statement](#)

Creates a decrypted copy of an existing database, including all transaction logs and dbspaces.

[CREATE ENCRYPTED DATABASE statement](#)

Creates an encrypted copy of an existing database, including all transaction logs and dbspaces; or creates a copy of an existing database with table encryption enabled.

[-gu database server option](#)

Sets the privilege required for executing database file administration statements such as for creating or dropping databases.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CREATE DOMAIN statement

Creates a domain in a database.

Syntax

```
CREATE { DOMAIN | DATATYPE } domain-name [ AS ] data-type  
[ [ NOT ] NULL ]  
[ DEFAULT default-value ]  
[ CHECK ( condition ) ]  
[ AS USER user-name ]
```

domain-name : identifier

data-type : built-in data type, with precision and scale, or another domain

Parameters

DOMAIN | DATATYPE clause It is recommended that you use CREATE DOMAIN, rather than CREATE DATATYPE, because CREATE DOMAIN is defined in the ANSI/ISO SQL Standard.

data-type Set the data type to one of the builtin data types or to the data type of another domain by specifying that domain name. For example:

```
CREATE DOMAIN a INT;  
CREATE DOMAIN b a;
```

You can also a %TYPE or %ROWTYPE attribute to set the data type to the data type of a column or row in a table or view. However, specifying a table reference variable for the %ROWTYPE (TABLE REF (*table-reference-variable*) %ROWTYPE) is now allowed.

NULL clause This clause allows you to specify the nullability of a domain. When a domain is used to define a column, nullability is determined as follows:

- Nullability specified in the column definition.
- Nullability specified in the domain definition.
- If the nullability was not explicitly specified in either the column definition or the domain definition, then the setting of the allow_nulls_by_default option is used.

CHECK clause When creating a domain with a CHECK constraint, you can use a variable name prefixed with the @ sign in the CHECK constraint's search condition. When the data type is used in the definition of a column, such a variable is replaced by the column name. This allows a domain's CHECK constraint to be applied to each table column defined with that domain.

AS USER clause Specifies the owner of the object.

Remarks

Domains are aliases for built-in data types, including precision and scale values where applicable. They improve convenience and encourage consistency in the database.

Domains are objects within the database. Their names must conform to the rules for identifiers. Domain names are always case insensitive, as are built-in data type names, but they are not collation-insensitive. For example, in the Turkish collation, the domain name "image" can be used

(because it was created using lowercase letters) but not the domain "IMAGE". The letter case that is guaranteed to work is the letter case in which the type was created and this can be seen by looking at the type_name column of the SYS.SYSUSERTYPE table.

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 domain name must be unique, and all users can access the data type without using the owner as prefix.

Domains can have CHECK conditions and DEFAULT values, and you can indicate whether the data type permits NULL values or not. These conditions and values are inherited by any column defined on the domain. Any conditions or values explicitly specified in the column definition override those specified for the domain.

Privileges

You must have the CREATE DATATYPE or CREATE ANY OBJECT system privilege to create domains owned by you. You cannot create domains owned by others.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Domain support is optional ANSI/ISO SQL Language Feature F251.

Example

The following statement creates a ROW domain named MyRow, which can hold a row of data:

```
CREATE DOMAIN MyRow ROW( a INT, b INT );
```

The new MyRow domain can then be referenced in SQL statements. For example:

```
CREATE FUNCTION Swap( @parm MyRow )
  RETURNS (MyRow)
  BEGIN
    DECLARE @ret MyRow;
    SET @ret = ROW( @parm.b, @parm.a );
    RETURN @ret;
  END;
```

The following statement creates an ARRAY domain named MyArray, which can hold an array of rows:

```
CREATE DOMAIN MyArray ARRAY( 10 ) OF ROW( a INT, b INT );
```

The following statement creates a domain named address, which holds a 35-character string, and which may be NULL.

```
CREATE DOMAIN address CHAR( 35 ) NULL;
```

The following statement creates a domain named ID, which does not allow NULLS, and which is set

to autoincrement by default.

```
CREATE DOMAIN ID INT
NOT NULL
DEFAULT AUTOINCREMENT;
```

The following statement creates a domain named `PhoneNumber`, which uses a regular expression within a `CHECK` constraint to ensure that the string has a properly formatted North American phone number of 12 characters, consisting of a 3-digit area code, 3-digit exchange, and 4-digit number separated by either dashes or a blank.

```
CREATE DOMAIN PhoneNumber CHAR(12) NULL
CHECK( @PhoneNumber REGEXP '([2-9][0-9]{2}-[2-9][0-9]{2}-[0-9]{4})|([2-9]
[0-9]{2}\s[2-9][0-9]{2}\s[0-9]{4})');
```

Some columns in a database are used for employee names and others to store addresses. You might then define the following domains.

```
CREATE DOMAIN persons_name CHAR(30)
CREATE DOMAIN street_address CHAR(35);
```

Having defined these domains, you can use them much as you would the built-in data types. You can use these definitions to define a table, as follows. You need the `CREATE TABLE` privilege to execute the following statement.

```
CREATE TABLE myCustomers (
    ID INT DEFAULT AUTOINCREMENT PRIMARY KEY,
    Name persons_name,
    Street street_address);
```

In the above example, the table's primary key is specified to be of type integer. Indeed, many of your tables may require similar identifiers. Instead of specifying that these are integers, it is much more convenient to create an identifier domain for use in these applications.

When you create a domain, you can specify a default value and provide check constraint to ensure that no inappropriate values are typed into any column of this type.

Integer values are commonly used as table identifiers. A good choice for unique identifiers is to use positive integers. Since such identifiers are likely to be used in many tables, you could define the following domain.

```
CREATE DOMAIN identifier UNSIGNED INT
DEFAULT AUTOINCREMENT;
```

Using this definition, you can rewrite the definition of the `Customers` table, shown above.

```
CREATE TABLE Customers2 (  
    ID identifier PRIMARY KEY,  
    Name persons_name,  
    Street street_address  
);
```

The second two CREATE DOMAIN statements in the following set of statements create domains based on the data types of the Surname and GivenName columns of the Customers table.

```
CREATE DOMAIN identifier UNSIGNED INT  
DEFAULT AUTOINCREMENT;  
CREATE DOMAIN customers_surname Customers.Surname%TYPE;  
CREATE DOMAIN customers_givenname Customers.GivenName%TYPE;  
  
CREATE TABLE Customers3 (  
    ID identifier PRIMARY KEY,  
    Surname customers_surname,  
    GivenName customers_givenname  
);
```

Related Information

[%TYPE and %ROWTYPE attributes](#)

In addition to explicitly setting the data type for an object, you can also set the data type by specifying the %TYPE and %ROWTYPE attributes.

[allow_nulls_by_default option](#)

Controls whether new columns that are created without specifying either NULL or NOT NULL are allowed to contain NULL values.

[DROP DOMAIN statement](#)

Removes a domain (data type) from the database.

[SQL data types](#)

There are many SQL data types supported by the software.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

CREATE ENCRYPTED DATABASE statement

Creates an encrypted copy of an existing database, including all transaction logs and dbspaces; or creates a copy of an existing database with table encryption enabled.

Syntax

- **Create an encrypted copy of a database**

```
CREATE ENCRYPTED DATABASE newfile
FROM oldfile
[ KEY newkey ]
[ ALGORITHM algorithm ]
[ OLD KEY oldkey ]
```

algorithm :

```
'SIMPLE'
| 'AES'
| 'AES256'
| 'AES_FIPS'
| 'AES256_FIPS'
```

- **Create a copy of a database with table encryption enabled**

```
CREATE ENCRYPTED TABLE DATABASE newfile
FROM oldfile
[ KEY newkey ]
[ ALGORITHM algorithm ]
[ OLD KEY oldkey ]
```

Parameters

CREATE ENCRYPTED DATABASE clause Specifies the name for the new database.

CREATE ENCRYPTED TABLE DATABASE clause Specifies the name for the new database. The new database is not encrypted, but has table encryption enabled.

FROM clause Specifies the name of the original database file (*oldfile*).

KEY clause Specifies the encryption key for *newfile*. The key can be either a string or a variable name. Not required for ALGORITHM 'SIMPLE'.

ALGORITHM clause Specifies the encoding algorithm to use for *newfile*. You may choose between SIMPLE obfuscation or some form of AES encryption. If you specify a KEY clause but do not specify the ALGORITHM clause, AES (128-bit encryption) is used by default. If you specify 'SIMPLE' for *algorithm*, you do not specify a KEY clause.

OLD KEY clause Use this clause to specify the encryption key for *oldfile*. The key can be either a string or a variable name. This clause is only required if *oldfile* is encrypted using some form of AES encryption.

Remarks

You can use this statement to create an encrypted copy of an existing database, including all transaction logs and dbspaces.

You can also use this statement to create a copy of a database and enable table encryption in the copy.

The database file *oldfile* can be an unencrypted database, an encrypted database, or a database with table encryption enabled.

Creating an encrypted copy of a database takes an existing database, *oldfile*, and creates an encrypted copy of it, *newfile*.

Creating a copy of a database with table encryption enabled takes an existing database, *oldfile*, and creates a copy of it, *newfile*, with table encryption enabled. When you use this syntax, any tables encrypted in *oldfile* are encrypted in *newfile* as well. If no tables were encrypted in *oldfile*, but you want to encrypt them, you can execute an ALTER TABLE...ENCRYPTED statement on each table you want to encrypt.

Neither syntax replaces or removes *oldfile*.

If *oldfile* uses transaction log or transaction log mirror files, they are renamed *newfile.log* and *newfile.mlg* respectively.

If *oldfile* contains dbspace files, an E (for encrypted) is added to the file name. For example, when you execute the CREATE ENCRYPTED DATABASE statement, the file *mydbspace.dbs* is changed to *mydbspace.dbsE*.

You can use this statement to change the encryption algorithm and key for a database. However, the CREATE ENCRYPTED DATABASE statement produces a new file (*newfile*), and does not replace or remove the previous version of the file (*oldfile*).

CREATE ENCRYPTED DATABASE and CREATE ENCRYPTED TABLE DATABASE cannot be executed against a database that requires recovery. These statements are not supported in procedures, triggers, events, or batches.

You cannot be connected to the database you are encrypting. You must be connected to a different database. For example, connect to the utility database. The database that you are encrypting must not be running.

You can also encrypt an existing database or change an existing encryption key by unloading and reloading the database using the dbunload -an option with either -ek or -ep.

You can also create an encrypted database, or a database with table encryption enabled, using the CREATE DATABASE statement.

If disk sandboxing is enabled, then database operations are limited to the directory where the main database file is located.

Privileges

Your ability to execute this statement depends on the setting for the -gu database option, and whether you have the SERVER OPERATOR system privilege.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example creates an encrypted copy of the sample database called *demoEnc.db*. The

new database is encrypted with AES256 encryption.

```
CREATE ENCRYPTED DATABASE 'demoEnc.db'  
FROM 'C:\\Users\\Public\\Documents\\SQL Anywhere  
17\\Samples\\C:\\Users\\Public\\Documents\\SQL Anywhere  
17\\Samples'  
KEY 'Sd8f6654*Mnn'  
ALGORITHM 'AES256';
```

The following example creates a copy of the sample database called *demoTableEnc.db*. Table encryption is enabled on the new database. Since a key was specified with no algorithm, AES encryption is used.

```
CREATE ENCRYPTED TABLE DATABASE 'demoTableEnc.db'  
FROM 'C:\\Users\\Public\\Documents\\SQL Anywhere  
17\\Samples\\C:\\Users\\Public\\Documents\\SQL Anywhere  
17\\Samples'  
KEY 'Sd8f6654';
```

Related Information

[Tips on rebuilding databases using the Unload utility](#)

You can use the Unload utility (dbunload) to unload an entire database into a text comma-delimited format and create the necessary SQL script files to completely recreate your database.

[Database encryption and decryption](#)

You can encrypt your database to make it more difficult for someone to decipher the data in your database.

[Table encryption](#)

Table encryption allows you to encrypt tables or materialized views with sensitive data, without the performance impact that encrypting the entire database might cause.

[Database encoding options](#)

Two methods of database encoding are supported: simple obfuscation and strong encryption.

[CREATE DECRYPTED DATABASE statement](#)

Creates a decrypted copy of an existing database, including all transaction logs and dbspaces.

[CREATE ENCRYPTED FILE statement](#)

Creates a strongly encrypted copy of a database file, transaction log, transaction log mirror, or dbspace.

[CREATE DECRYPTED FILE statement](#)

Creates a decrypted copy of a strongly encrypted database, and can be used to create decrypted copies of transaction logs, transaction log mirrors, and dbspaces.

[CREATE DATABASE statement](#)

Creates a database.

[ALTER TABLE statement](#)

Modifies a table definition or disables dependent views.

[Initialization utility \(dbinit\)](#)

Creates a new database.

[-gu database server option](#)

Sets the privilege required for executing database file administration statements such as for creating or dropping databases.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

CREATE ENCRYPTED FILE statement

Creates a strongly encrypted copy of a database file, transaction log, transaction log mirror, or dbspace.

Syntax

```
CREATE ENCRYPTED FILE newfile  
FROM oldfile  
KEY newkey  
[ ALGORITHM algorithm ]  
[ OLD KEY oldkey ]
```

```
algorithm :  
  'AES'  
  | 'AES256'  
  | 'AES_FIPS'  
  | 'AES256_FIPS'
```

Parameters

FROM clause Specifies the name of the original database file (*oldfile*).

KEY clause Specifies the encryption key to use for *newfile*. The key can be either a string or a variable name. This key must be specified.

ALGORITHM clause Specifies the algorithm used to encrypt *newfile*. If you do not specify an algorithm, AES (128-bit encryption) is used by default.

OLD KEY clause Specifies the encryption key for *oldfile*, if it is encrypted. The key can be either a string or a variable name.

Remarks

Use this statement when your database requires recovery and you need to create an encrypted copy of the database for support reasons. You must also use this statement to encrypt any database-related files such as the transaction log, transaction log mirror, or dbspace files.

You cannot be connected to the database you are creating the encrypted file for. You must be connected to a different database. For example, connect to the utility database. The database that you are encrypting must not be running.

When encrypting the database-related files, you must specify the same algorithm and key for all files related to the database.

If *oldfile* has dbspaces or transaction log files associated with it and you encrypt those too, you must ensure that the new name and location of those files is stored with the new database. To do so:

- Run `dblog -t` on the new database to change the name and location of the transaction log.
- Run `dblog -m` on the new database to change the name and location of the transaction log mirror.
- Execute an `ALTER DBSPACE` statement on the new database to change the location and name of the dbspace files.

If disk sandboxing is enabled, then database operations are limited to the directory where the main database file is located.

You can use this statement to change the encryption algorithm and key for a database. However, the CREATE ENCRYPTED FILE statement produces a new file (*newfile*), and does not replace or remove the previous version of the file (*oldfile*).

The name of the transaction log file remains the same in this process, so if the database and transaction log file are renamed, then you need to run dblog -t on the resulting database.

You can also encrypt an existing database or change an existing encryption key by unloading and reloading the database using the dbunload -an option with either -ek or -ep.

If you have a database on which table encryption is enabled, you cannot encrypt the database using this statement. However, you can use this statement to change the key used for table encryption. To encrypt a database that has table encryption enabled, use the CREATE ENCRYPTED DATABASE statement.

This statement is not supported in procedures, triggers, events, or batches.

Privileges

Your ability to execute this statement depends on the setting for the -gu database option, and whether you have the SERVER OPERATOR system privilege.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example encrypts the sample database *demo.db* and creates a new database called *demo2.db* that is encrypted with AES_FIPS encryption. The new database file is placed in the server's current working directory.

```
CREATE ENCRYPTED FILE 'demo2.db'  
FROM 'C:\\Users\\Public\\Documents\\SQL Anywhere  
      17\\Samples\\demo.db'  
KEY 'Sd8f6654*Mnn'  
ALGORITHM 'AES_FIPS';
```

The following example encrypts the sample database *demo.db* and its transaction log file *demo.log*.

```
CREATE ENCRYPTED FILE 'demo3.db'  
FROM 'C:\\Users\\Public\\Documents\\SQL Anywhere  
      17\\Samples\\demo.db'  
KEY 'Sd8f6654*Mnn';  
  
CREATE ENCRYPTED FILE 'demo3.log'  
FROM 'C:\\Users\\Public\\Documents\\SQL Anywhere  
      17\\Samples\\demo.log'  
KEY 'Sd8f6654*Mnn';
```

The new database file and transaction log are placed in the server's current working directory. At a command prompt, use the Transaction Log utility (dblog) to set the new transaction log name since the new database file *demo3.db* still references the old transaction log file.

```
dblog -ek Sd8f6654*Mnn -t demo3.log demo3.db
```

To change the encryption key for a database, create a copy of the database file and transaction log using the new key, as shown in the following example:

```
CREATE ENCRYPTED FILE '\temp\demo.db'  
FROM 'C:\\Users\\Public\\Documents\\SQL Anywhere  
17\\Samples\\demo.db'  
KEY 'Sd251072*Mnn'  
OLD KEY 'Sd8f6654*Mnn';  
  
CREATE ENCRYPTED FILE '\temp\demo.log'  
FROM 'C:\\Users\\Public\\Documents\\SQL Anywhere  
17\\Samples\\demo.log'  
KEY 'Sd251072*Mnn'  
OLD KEY 'Sd8f6654*Mnn';
```

The new database file and transaction log are placed in the specified directory. Now you can archive the old database file and its transaction log, and then move the new database file and transaction log to the same directory where the old files were located.

Related Information

[Database encryption and decryption](#)

You can encrypt your database to make it more difficult for someone to decipher the data in your database.

[CREATE ENCRYPTED DATABASE statement](#)

Creates an encrypted copy of an existing database, including all transaction logs and dbspaces; or creates a copy of an existing database with table encryption enabled.

[CREATE DECRYPTED FILE statement](#)

Creates a decrypted copy of a strongly encrypted database, and can be used to create decrypted copies of transaction logs, transaction log mirrors, and dbspaces.

[CREATE DECRYPTED DATABASE statement](#)

Creates a decrypted copy of an existing database, including all transaction logs and dbspaces.

[Unload utility \(dbunload\)](#)

Unloads a database into a specified directory.

[Transaction Log utility \(dblog\)](#)

Administers the transaction log for a database.

[-gu database server option](#)

Sets the privilege required for executing database file administration statements such as for creating or dropping databases.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)» [Alphabetical list of SQL statements](#)

CREATE EVENT statement

Defines an event and its associated handler for automating predefined actions, and to define scheduled actions.

Syntax

```

CREATE [ OR REPLACE ] EVENT [user-name.]event-name
[ TYPE event-type
  [ WHERE trigger-condition [ AND trigger-condition ] ... ]
  | SCHEDULE schedule-spec, ... ]
[ ENABLE | DISABLE ]
[ AT { CONSOLIDATED | REMOTE | ALL } ]
[ FOR { PRIMARY | ALL } ]
[ HANDLER
  BEGIN
...
  END ]

```

event-type :

```

BackupEnd
| Connect
| ConnectFailed
| DatabaseStart
| DBDiskSpace
| Deadlock
| "Disconnect"
| GlobalAutoincrement
| GrowDB
| GrowLog
| GrowTemp
| LogDiskSpace
| MirrorFailover
| MirrorServerDisconnect
| RAISERROR
| ServerIdle
| TempDiskSpace

```

trigger-condition :

```

event_condition( condition-name ) {
=
| <
| >
| !=
| <=
| >=
} value

```

```

schedule-spec :
[ schedule-name ]
  { START TIME start-time | BETWEEN start-time AND end-time }
  [ EVERY period { HOURS | MINUTES | SECONDS } ]
  [ ON { ( day-of-week, ... ) | ( day-of-month, ... ) } ]
  [ START DATE start-date ]

```

event-name : *identifier*

schedule-name : *identifier*

day-of-week : *string*

day-of-month : *integer*

value : *integer*

period : *integer*

start-time : *time*

end-time : *time*

start-date : *date*

Parameters

CREATE EVENT clause The event name is an identifier. An event has a creator, which is the user creating the event, and the event handler executes with the privileges of that creator. This is the same as stored procedure execution. You cannot create events owned by other users.

user-name Optionally, specify the name of a user in the system; when the event runs, it runs with the privileges of *user-name*. If this parameter is not specified, the event runs with the privileges of the user who created the event. *user-name* should not be confused with an owner of the event, however; events do not have owners.

OR REPLACE clause Specifying OR REPLACE (CREATE OR REPLACE EVENT) creates an event or replaces an event with the same name. If the event already exists, then all comments are preserved when you use the OR REPLACE clause, but all existing attributes of the event are dropped.

TYPE clause You can specify the TYPE clause with an optional WHERE clause, or specify the SCHEDULE.

The *event-type* is one of the listed set of system-defined event types. The event types are case insensitive. To specify the conditions under which this *event-type* triggers the event, use the WHERE clause.

- **DiskSpace event types** If the database contains an event handler for one of the DiskSpace types, the database server checks the available space on each device associated with the relevant file every 30 seconds.

In the event the database has more than one dbspace, on separate drives, DBDiskSpace checks each drive and acts depending on the lowest available space.

The LogDiskSpace event type checks the location of the transaction log and any

transaction log mirror, and reports based on the least available space.

The TempDiskSpace event type checks the amount of temporary disk space.

If the appropriate event handlers have been defined (DBDiskSpace, LogDiskSpace, or TempDiskSpace), the database server checks the available space on each device associated with a database file every 30 seconds. Similarly, if an event has been defined to handle the system event type ServerIdle, the database server notifies the handler when no requests have been processed during the previous 30 seconds.

You can specify the -fc option when starting the database server to implement a callback function when the database server encounters a file system full condition.

- **GlobalAutoincrement event type** The event fires on *each* insert when the number of remaining values for a GLOBAL AUTOINCREMENT is less than 1% of the end of its range. A typical action for the handler could be to request a new value for the global_database_id option, based on the table and number of remaining values which are supplied as parameters to this event.

You can use the event_condition function with RemainingValues as an argument for this event type.

- **ServerIdle event type** If the database contains an event handler for the ServerIdle type, the database server checks for server activity every 30 seconds.
- **Database mirroring event types** The MirrorServerDisconnect event fires when a connection from the primary database server to the mirror server or arbiter server is lost, and the MirrorFailover event fires whenever a server takes ownership of the database.

WHERE clause The trigger condition determines the condition under which an event is fired. For example, to take an action when the disk containing the transaction log becomes more than 80% full, use the following triggering condition:

```
...
WHERE event_condition( 'LogFreePercent' ) < 20
...
```

The argument to the event_condition function must be valid for the event type.

You can use multiple AND conditions to make up the WHERE clause, but you cannot use OR conditions or other conditions.

You can specify a variable name for the event_condition value.

SCHEDULE clause This clause specifies when scheduled actions are to take place. The sequence of times acts as a set of triggering conditions for the associated actions defined in the event handler.

You can create more than one schedule for a given event and its associated handler. This permits complex schedules to be implemented. You must provide a *schedule-name* when there is more than one schedule; the *schedule-name* is optional if you provide only a single schedule.

A scheduled event is recurring if its definition includes EVERY or ON; if neither of these reserved words is used, the event executes at most once. An attempt to create a non-recurring scheduled event for which the start time has passed generates an error. When a non-recurring scheduled event has passed, its schedule is deleted, but the event handler is not deleted.

Scheduled event times are calculated when the schedules are created, and again when the event handler completes execution. The next event time is computed by inspecting the schedule

or schedules for the event, and finding the next schedule time that is in the future. If an event handler is instructed to run every hour between 9:00 and 5:00, and it takes 65 minutes to execute, it runs at 9:00, 11:00, 1:00, 3:00, and 5:00. If you want execution to overlap, you must create more than one event.

The subclauses of a schedule definition are as follows:

- **START TIME clause** The first scheduled time for each day on which the event is scheduled. The *start-time* parameter is a string, and cannot be an expression such as `NOW()`. If a START DATE is specified, the START TIME refers to that date and each subsequent day (if the schedule includes EVERY or ON). If no START DATE is specified, the START TIME is on the current day (unless the time has passed) and each subsequent day (if the schedule includes EVERY or ON). The clause START TIME *start-time* is equivalent to BETWEEN *start-time* AND '23:59:59'.

You can specify a variable name for *start-time*.

- **BETWEEN...AND clause** A range of times during the day outside which no scheduled times occur. The *start-time* and *end-time* parameters are strings, and cannot be expressions such as `NOW()`. If a START DATE is specified, the scheduled times do not occur until that date.

You can specify a variable name for *start-time* and *end-time*.

- **EVERY clause** An interval between successive scheduled events. Scheduled events occur only after the START TIME for the day, or in the range specified by BETWEEN...AND.

You can specify a variable name for *period*.

- **ON clause** A list of days on which the scheduled events occur. The default is every day if EVERY is specified. Days can be specified as days of the week or days of the month.

Days of the week are Mon, Tues, and so on. You may also use the full forms of the day, such as Monday. You must use the full forms of the day names if the language you are using is not English, is not the language requested by the client in the connection string, and is not the language which appears in the database server messages window.

Days of the month are integers from 0 to 31. A value of 0 represents the last day of any month.

- **START DATE clause** The date on which scheduled events are to start occurring. This value is a string, and cannot be an expression such as `TODAY()`. The default is the current date.

You can specify a variable name for *start-date*.

Each time a scheduled event handler is completed, the following actions are taken to calculate the next scheduled time and date for the event:

1. If the EVERY clause is used, find whether the next scheduled time falls on the current day, and is before the end time specified by the BETWEEN...AND clause, if it was specified. If so, that is the next scheduled time.
2. If the next scheduled time does not fall on the current day, find the next date on which the event is to be executed and use the START TIME for that date, or the beginning of the BETWEEN...AND range.

- **ENABLE | DISABLE clause** By default, event handlers are enabled. When DISABLE is specified, the event handler does not execute even when the scheduled time or triggering condition occurs. A TRIGGER EVENT statement does *not* cause a disabled event handler to be executed.

- **AT clause** This clause should be used only in the following circumstance: in a SQL Remote

setup, use the AT clause against your remote or consolidated databases to restrict the databases at which the event is handled.

If you do not use the AT clause when creating events for SQL Remote, all databases execute the event. When executed on a consolidated database, this statement does not affect remote databases that have already been extracted.

- **FOR clause** This clause should only be used in the following circumstance: in a database mirroring or read-only scale-out system, use the FOR clause to restrict the databases at which the event is handled.

If you do not use the FOR clause when creating an event for a database in a mirroring or read-only scale-out system, then only the database that is running on the primary server executes the event. The following subclauses are supported:

- **FOR PRIMARY** The event executes only on the server currently acting as the primary server. The default is the PRIMARY sub clause.

When the FOR PRIMARY clause (or the FOR clause is not specified) is used with the DatabaseStart event type, the event executes when a server becomes the primary server for the database.

- **FOR ALL** The event executes on all servers in the system.

When the FOR ALL clause is used with the DatabaseStart event type, the event is executed when any database starts. If in a mirroring system the event did not run when the database started (for example, the database was running before the event was created), then the event can execute during a fail over. For example you start a database mirroring system, you create a DatabaseStart event with the FOR ALL clause, and then you stop the primary server, which causes a fail over. In this example, the event executes on the new primary server. The DatabaseStart event will not execute during subsequent fail overs.

- **HANDLER clause** Each event has one handler.

Remarks

Events can be used for:

- **Scheduling actions** The database server executes actions on a timed schedule. You can use this capability to complete scheduled tasks such as backups, validity checks, and queries used to add data to reporting tables.
- **Event handling actions** The database server executes actions when a predefined event occurs. You can use this capability to complete scheduled tasks such as restrict disk space when a disk fills beyond a specified percentage. Event handler actions are committed if errors are not detected during execution, and rolled back if errors are detected.

An event definition includes two distinct pieces. The trigger condition can be an occurrence, such as a disk filling up beyond a defined threshold. A schedule is a set of times, each of which acts as a trigger condition. When a trigger condition is satisfied, the event handler executes. The event handler includes one or more actions specified inside a compound statement (BEGIN... END).

If no trigger condition or schedule specification is supplied, only an explicit TRIGGER EVENT statement can trigger the event. During development, you may want to test event handlers using TRIGGER EVENT, and add the schedule or WHERE clause once testing is complete.

Event errors are logged to the database server message log.

After each execution of an event handler, a COMMIT occurs if no errors occurred. A ROLLBACK occurs if there was an error.

When event handlers are triggered, the database server makes context information, such as the connection ID that caused the event to be triggered, available to the event handler using the `event_parameter` function.

Event handlers execute on a separate connection, but the separate connection does not count towards the ten-connection limit of the personal database server.

An owner can be specified but is ignored; events do not have owners. Because events do not have owners, no two events can have the same name.

Note For parameters that accept variable names, an error is returned if one of the following conditions is true:

- The variable does not exist
- The contents of the variable are NULL
- The variable exceeds the length allowed by the parameter
- The data type of the variable does not match that required by the parameter

Privileges

If you are creating a new event, then you must have the `MANAGE ANY EVENT` or `CREATE ANY OBJECT` system privilege.

If you are replacing an existing event, then you must have one of the following:

- `MANAGE ANY EVENT` system privilege
- `ALTER ANY OBJECT` system privilege
- `CREATE ANY OBJECT` and `DROP ANY OBJECT` system privileges

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

Instruct the database server to perform an automatic backup to tape using the first tape drive, every day at 1 A.M.

```
CREATE EVENT DailyBackup
  SCHEDULE daily_backup
  START TIME '1:00AM' EVERY 24 HOURS
  HANDLER
    BEGIN
      BACKUP DATABASE TO '\\\\.\tape0'
      ATTENDED OFF
    END;
```

Instruct the database server to perform an incremental backup daily at 1:00 A.M.

```

CREATE EVENT IncrementalBackup
SCHEDULE
  START TIME '1:00 AM' EVERY 24 HOURS
HANDLER
BEGIN
  BACKUP DATABASE DIRECTORY 'c:\\backup'
  TRANSACTION LOG ONLY
  TRANSACTION LOG RENAME MATCH
END;

```

Instruct the database server to perform an automatic backup of the transaction log only, every hour, Monday to Friday between 8 A.M. and 6 P.M.

```

CREATE EVENT HourlyLogBackup
SCHEDULE hourly_log_backup
BETWEEN '8:00AM' AND '6:00PM'
EVERY 1 HOURS ON
  ('Monday','Tuesday','Wednesday','Thursday','Friday')
HANDLER
BEGIN
  BACKUP DATABASE DIRECTORY 'c:\\database\\backup'
  TRANSACTION LOG ONLY
  TRANSACTION LOG RENAME
END;

```

Determine when an event is next scheduled to run:

```

SELECT DB_EXTENDED_PROPERTY( 'NextScheduleTime', 'HourlyLogBackup');

```

The following statements create three variables, one for start time, one for end time, and one for interval time, and then creates an event using variables for the START TIME and EVERY clauses.

```

CREATE VARIABLE @st1 LONG VARCHAR
CREATE VARIABLE @int1 INTEGER
SET @st1 = ' 2:00AM '
SET @int1 = 12;

CREATE EVENT DailyBackup
SCHEDULE daily_backup
START TIME @st1 EVERY @int1 HOURS
HANDLER
BEGIN
  BACKUP DATABASE TO '\\\\.\\tape0'
  ATTENDED OFF
END;

```

The following example creates an event that uses a variable for one of the event_condition values, and then creates an event that uses the variable @i1 for the first event_condition value:

```
CREATE VARIABLE @i1 INTEGER;
SET @i1 = 10000;

CREATE EVENT LogNotifier
TYPE RaiseError
WHERE event_condition ( 'ErrorNumber' ) <> @i1 AND event_condition (
'ErrorNumber' ) <> 7
HANDLER
BEGIN
    MESSAGE 'LogNotifier message'
END;
```

Related Information

[System events](#)

Each system event provides a hook on which you can program a set of actions.

[Trigger conditions for events](#)

Each event definition has a system event associated with it, and one or more trigger conditions.

[Database server logging](#)

The **database server message log** contains informational messages, errors, warnings, and messages from the MESSAGE statement.

[BEGIN statement](#)

Specifies a compound statement.

[ALTER EVENT statement](#)

Changes the definition of an event or its associated handler for automating predefined actions, or alters the definition of scheduled actions. You can also use this statement to hide the definition of an event handler.

[TRIGGER EVENT statement](#)

Triggers a named event. The event may be defined for event triggers or be a scheduled event.

[EVENT_PARAMETER function \[System\]](#)

Provides context information for event handlers.

[EVENT_CONDITION function \[System\]](#)

Specifies when an event handler is triggered.

[Troubleshooting: Using database mirroring system events to send notification email when failover occurs](#)

Use MirrorServerDisconnect and the MirrorFailover events to send email notification that an action may be required on the mirror database.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CREATE EXISTING TABLE statement

Creates a new proxy table, which represents an existing object on a remote server.

Syntax

```
CREATE { EXISTING | VIRTUAL } TABLE [owner.]table-name  
[ column-definition, ... ]  
AT location-string
```

column-definition :
column-name *data-type* **NOT NULL**

location-string :
remote-server-name.*[db-name]*.*[owner]*.*object-name*
| *remote-server-name*;*[db-name]*;*[owner]*;*object-name*

Parameters

CREATE { EXISTING | VIRTUAL } TABLE CREATE EXISTING TABLE and CREATE VIRTUAL TABLE are semantically equivalent. CREATE VIRTUAL TABLE is provided for compatibility with SAP HANA.

AT clause The AT clause specifies the location of the remote object. The AT clause supports the semicolon (;) as a delimiter. If a semicolon is present anywhere in the *location-string* string, then the semicolon is the field delimiter. If no semicolon is present, then a period is the field delimiter. This behavior allows file names and extensions to be used in the database and owner fields.

The string in the AT clause can contain local or global variable names enclosed in braces (for example, {*variable-name*}). The SQL variable name must be of type CHAR, VARCHAR, or LONG VARCHAR. For example, an AT clause that contains '*access*;*{@myfile}*;;*a1*' indicates that *@myfile* is a SQL variable and that the current contents of the *@myfile* variable should be substituted when the proxy table is created.

Remarks

The CREATE EXISTING TABLE statement creates a new, local, proxy table that maps to a table at an external location. The CREATE EXISTING TABLE statement is a variant of the CREATE TABLE statement. The EXISTING keyword is used with CREATE TABLE to specify that a table already exists remotely and to import its metadata. This syntax establishes the remote table as a visible entity to users. The software verifies that the table exists at the external location before it creates the table.

If the object does not exist (either as a host data file or remote server object), the statement is rejected with an error message.

Index information from the host data file or remote server table is extracted and used to create rows for the ISYSIDX system table. This information defines indexes and keys in server terms and enables the query optimizer to consider any indexes that may exist on this table.

Referential constraints are passed to the remote location when appropriate.

If *column-definitions* are not specified, then the database server derives the column list from the metadata it obtains from the remote table. If *column-definitions* are specified, then the database

server verifies the *column-definitions*. Column names, data types, lengths, the identity property, and null properties are checked for the following conditions:

- Column names must match identically (although case is ignored).
- Data types in the CREATE EXISTING TABLE statement must match or be convertible to the data types of the column on the remote location. For example, a local column data type is defined as money, while the remote column data type is numeric.
- Each column's NULL property is checked. If the local column's NULL property is not identical to the remote column's NULL property, then a warning message is issued, but the statement is not aborted.
- Each column's length is checked. If the length of CHAR, VARCHAR, BINARY, VARBINARY, DECIMAL and/or NUMERIC columns do not match, then a warning message is issued, but the command is not aborted.

You may choose to include only a subset of the actual remote column list in your CREATE EXISTING statement.

Privileges

You must have the CREATE PROXY TABLE system privilege to create proxy tables owned by you. You must have the CREATE ANY TABLE or CREATE ANY OBJECT system privilege to create proxy tables owned by others.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.
- **Transact-SQL** Supported by Adaptive Server Enterprise. The format of *location-string* is implementation-defined.

Example

Create a proxy table named blurbs for the blurbs table at the remote server server_a.

```
CREATE EXISTING TABLE blurbs
( author_id ID not null,
  copy text not null)
AT 'server_a.db1.joe.blurbs';
```

Create a proxy table named blurbs for the blurbs table at the remote server server_a. The database server derives the column list from the metadata it obtains from the remote table.

```
CREATE EXISTING TABLE blurbs
AT 'server_a.db1.joe.blurbs';
```

Create a proxy table named rda_employees for the Employees table at the remote server rda.

```
CREATE EXISTING TABLE rda_employees
AT 'rda...Employees';
```

Create a proxy table named `rda_employees` for a table that is specified by the SQL variable `table_name` at the remote server `rda`.

```
CREATE EXISTING TABLE rda_employees  
AT 'rda...{table_name}';
```

Related Information

[Proxy table locations](#)

Use the `AT` clause of the `CREATE TABLE` and the `CREATE EXISTING TABLE` statements to define the location of an existing object.

[CREATE TABLE statement](#)

Creates a new table in the database and, optionally, creates a table on a remote server.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

CREATE EXTERNLOGIN statement

Assigns an alternate login name and password to be used when communicating with a remote server.

Syntax

- **Create external login for a remote server**

```
CREATE EXTERNLOGIN login-name
TO remote-server
[ REMOTE LOGIN remote-user [ IDENTIFIED BY remote-password ] ]
```

- **Create external login for a remote server (include variables in syntax)**

```
CREATE EXTERNLOGIN USER string | variable
SERVER string | variable
[ REMOTE USER string | variable [ IDENTIFIED BY string | variable ] ]
```

- **Create external login for a directory access server**

```
CREATE EXTERNLOGIN login-name
TO remote-server
```

- **Create external login for a directory access server (include variables in syntax)**

```
CREATE EXTERNLOGIN USER string | variable
SERVER string | variable
```

Parameters

login-name Specifies the local user login name. When using integrated logins, *login-name* is the database user to which the Windows user or group is mapped.

TO clause The TO clause specifies the name of the remote server.

REMOTE LOGIN clause The REMOTE LOGIN clause specifies the user account on remote-server for the local user login-name. Values for the REMOTE LOGIN clause are restricted to 128 bytes.

user-name Specifies the database user name. For remote servers, when using integrated logins, the *user-name* is the database user to which the Windows user or group is mapped. This value can be a string or a variable.

SERVER clause Specify the name of the remote server or the directory access server.

REMOTE USER clause (remote server) The REMOTE USER clause specifies the user account on the remote server for the database user name. Values for the REMOTE USER clause are restricted to 128 bytes.

- **IDENTIFIED BY clause** Specify the remote password for the remote user. The remote user and remote password combination must be valid on the remote server. This clause applies only to remote servers, not to directory access servers.

If you omit the IDENTIFIED BY clause, then the password is sent to the remote server as NULL. However, if you specify IDENTIFIED BY "" (an empty string), then the password sent

is the empty string.

Remarks: Remote servers

CREATE EXTERNLOGIN assigns an alternate login name and password to be used when communicating with a remote server.

Connections to a remote server are first attempted using the current executing user's external login. If this user does not have an external login, then the connection is attempted using the DEFAULT LOGIN credentials. If the remote server was created without a DEFAULT LOGIN, and no external login has been defined for the user, then the connection is attempted with the current executing user's ID and password.

The REMOTE LOGIN clause is required only when the remote server requires a user ID and password for the connection. Having an external login without a remote login allows the DBA to control who can access the remote server and tells the remote access layer that logging in to the remote server does not require a user ID and password.

The password is stored internally in encrypted form. The *remote-server* must be known to the local server by an entry in the ISYSSERVER table.

Sites with automatic password expiration should plan for periodic updates of passwords for external logins.

CREATE EXTERNLOGIN cannot be used from within a transaction.

If you use this statement in a procedure, do not specify the password (IDENTIFIED BY clause) as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

Note For *required* parameters that accept variable names, the database server returns an error if any of the following conditions is true:

- The variable does not exist
- The contents of the variable are NULL
- The variable exceeds the length allowed by the parameter
- The data type of the variable does not match that required by the parameter

Remarks (directory access servers)

By default, database users must have external logins to access the directory access server. However, you can configure the directory access server to remove this requirement by creating a default external login that all users can use.

CREATE EXTERNLOGIN assigns an external login to be used when accessing a directory access server.

CREATE EXTERNLOGIN cannot be used from within a transaction.

Privileges

You must have the MANAGE ANY USER system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

This fictitious example maps a local user, DBA, to user sa with password Plankton when connecting to the server server1.

```
CREATE EXTERNLOGIN DBA
TO server1
REMOTE LOGIN sa
IDENTIFIED BY Plankton;
```

Related Information

[Creating external logins \(SQL Central\)](#)

Create an external login for a user to use to communicate with a remote server or a directory access server.

[DROP EXTERNLOGIN statement](#)

Drops an external login from the database.

[CREATE SERVER statement](#)

Creates a remote server or a directory access server.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

CREATE FUNCTION statement [External call]

Creates an interface to a native or external function.

Syntax

```
CREATE [ OR REPLACE ] FUNCTION [ owner.]function-name
( [ parameter, ... ] )
RETURNS data-type
[ SQL SECURITY { INVOKER | DEFINER } ]
[ [ NOT ] DETERMINISTIC ]
{ EXTERNAL NAME 'native-call'
  | EXTERNAL NAME 'c-call' LANGUAGE { C_ESQL32 | C_ESQL64 | C_ODBC32 |
C_ODBC64 }
  | EXTERNAL NAME 'clr-call' LANGUAGE CLR
  | EXTERNAL NAME 'perl-call' LANGUAGE PERL
  | EXTERNAL NAME 'php-call' LANGUAGE PHP
  | EXTERNAL NAME 'java-call' LANGUAGE JAVA
  | EXTERNAL NAME 'js-call' LANGUAGE JS }
```

parameter :

```
[ IN ] parameter-name data-type [ DEFAULT expression ]
```

result-column :

```
column-name data-type
```

native-call :

```
[ system-configuration:]function-name@library-file-prefix[.{ so | dll } ]
```

system-configuration :

```
{ generic-operating-system | specific-operating-system } [ ( processor-architecture ) ]
```

generic-operating-system :

```
{ Unix | Windows }
```

specific-operating-system :

```
{ AIX | HPUX | Linux | OSX | Solaris | WindowsNT }
```

processor-architecture :

```
{ 32 | 64 | ARM | IA64 | PPC | SPARC | X86 | X86_64 }
```

c-call :

```
[ operating-system:]function-name@library; ...
```

operating-system :

```
Unix
```

clr-call :

```
dll-name::function-name( param-type-1[, ... ] )
```

*perl-call :***<file=perl-file> \$sa_perl_return = perl-subroutine(\$sa_perl_arg0[, ...])***php-call :***<file=php-file> print php-func(\$argv[1][, ...])***java-call :***[package-name.]class-name.method-name java-method-signature***java-method-signature :***([java-field-descriptor, ...]) java-return-descriptor***java-field-descriptor and java-return-descriptor :*

```

{ Z
  | B
  | S
  | I
  | J
  | F
  | D
  | C
  | V
  | [descriptor
  | Lclass-name;
}

```

*js-call :***<js-return-descriptor><file=js-object> js-func(js-field-descriptor[...])***js-field-descriptor and js-return-descriptor :*

```

{ S
  | B
  | I
  | U
  | D
  | [descriptor
}

```

Parameters

Parameter names must conform to the rules for database identifiers. They must have a valid SQL data type, and must be prefixed by the keyword IN, signifying that the argument is an expression that provides a value to the function. However, function parameters are IN by default.

When functions are executed, then not all parameters need to be specified. If a DEFAULT value is provided in the CREATE FUNCTION statement, missing parameters are assigned the default values. If an argument is not provided by the caller and no default is set, then an error is given.

OR REPLACE clause Specifying CREATE OR REPLACE FUNCTION creates a new function, or replaces an existing function with the same name. This clause changes the definition of the function, but preserves existing privileges.

RETURNS clause Use the RETURNS clause to specify the data type for the result of the

function.

If a RETURNS clause is specified, then it must be the first clause of the statement.

SQL SECURITY clause The SQL SECURITY clause defines whether the function is executed as the INVOKER (the user who is calling the function), or as the DEFINER (the user who owns the function). The default is DEFINER. For external calls, this clause establishes the ownership context for unqualified object references in the external environment.

When SQL SECURITY INVOKER is specified, more memory is used because annotation must be done for each user that calls the function. Also, when SQL SECURITY INVOKER is specified, name resolution is done as the invoker as well. Therefore, qualify all object names (tables, procedures, and so on) with their appropriate owner. For example, suppose user1 creates the following function:

```
CREATE FUNCTION user1.myFunc()  
  RETURNS INT  
  SQL SECURITY INVOKER  
  BEGIN  
    DECLARE res INT;  
    SELECT COUNT(*) INTO res FROM table1;  
    RETURN res;  
  END;
```

If user2 attempts to run this function and a table user2.table1 *does not* exist, a table lookup error results. Additionally, if a user2.table1 *does* exist, that table is used instead of the intended user1.table1. To prevent this situation, qualify the table reference in the statement (user1.table1, instead of just table1).

[NOT] DETERMINISTIC clause Use this clause to indicate whether functions are deterministic or non-deterministic. If this clause is omitted, then the deterministic behavior of the function is unspecified (the default).

If a function is declared as DETERMINISTIC, then it should return the same value every time it is invoked with the same set of parameters.

If a function is declared as NOT DETERMINISTIC, then it is not guaranteed to return the same value for the same set of parameters. A function declared as NOT DETERMINISTIC is re-evaluated each time it is called in a query. This clause is required when it is known that the function result for a given set of parameters can vary.

Also, functions that have side effects such as modifying the underlying data should be declared as NOT DETERMINISTIC. For example, a function that generates primary key values and is used in an INSERT...SELECT statement should be declared NOT DETERMINISTIC:

```

CREATE FUNCTION keygen( increment INTEGER )
RETURNS INTEGER
NOT DETERMINISTIC
BEGIN
    DECLARE keyval INTEGER;
    UPDATE counter SET x = x + increment;
    SELECT counter.x INTO keyval FROM counter;
    RETURN keyval
END
INSERT INTO new_table
SELECT keygen(1), ...
FROM old_table;

```

Functions can be declared as DETERMINISTIC if they always return the same value for given input parameters.

EXTERNAL NAME *native-call* clause The EXTERNAL NAME clause is not supported for TEMPORARY functions.

A function using the EXTERNAL NAME clause with no LANGUAGE attribute defines an interface to a native function written in a programming language such as C. The native function is loaded by the database server into its address space.

Because *native-call* can specify multiple sets of operating systems, processors, libraries, and functions, more-precisely specified configurations take precedence over less-precisely defined configurations. For example, `Solaris(X86_64):myfunc64@mylib.so` takes precedence over `Solaris:myfunc64@mylib.so`.

For syntaxes that support *system-configuration*, if you do not specify *system-configuration*, then it is assumed that the procedure runs on all system configurations. Unix represents the following Unix-based operating systems: AIX, HP-UX, Linux, Mac OS X, and Solaris. Windows represents all versions of the Windows operating system.

The *specific-operating-system* and *processor-architecture* values are those operating systems and processors supported by SQL Anywhere server.

The library name (*library-file-prefix*) is followed by the file extension which is typically `.dll` on Windows and `.so` on Unix. In the absence of the extension, the software appends the platform-specific default file extension for libraries. For example:

```

CREATE FUNCTION mystring( IN instr LONG VARCHAR )
RETURNS LONG VARCHAR
EXTERNAL NAME 'mystring@mylib.dll;Unix:mystring@mylib.so';

```

A simpler way to write the EXTERNAL NAME clause, using platform-specific defaults, is as follows:

```

CREATE FUNCTION mystring( IN instr LONG VARCHAR )
RETURNS LONG VARCHAR
EXTERNAL NAME 'mystring@mylib';

```

When called, the library containing the function is loaded into the address space of the database server behavior. The native function executes as part of the database server. In this case, if the

function causes a fault, then the database server shuts down. Because of this behavior, loading and executing functions in an external environment using the LANGUAGE attribute is recommended. If a function causes a fault in an external environment, then the database server continues to run.

EXTERNAL NAME *c-call* clause To call a compiled native C function in an external environment instead of within the database server, the stored procedure or function is defined with the EXTERNAL NAME clause followed by the LANGUAGE attribute.

When the LANGUAGE attribute is specified, then the library containing the function is loaded by an external process and the external function executes as part of that external process. In this case, if the function causes a fault, then the database server continues to run.

```
CREATE FUNCTION ODBCinsert(
    IN ProductName CHAR(30),
    IN ProductDescription CHAR(50)
)
RETURNS INT
EXTERNAL NAME 'ODBCexternalInsert@extodbc.dll'
LANGUAGE C_ODBC32;
```

EXTERNAL NAME *clr-call* clause To call a .NET function in an external environment, the function interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE CLR attribute.

A CLR stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in a .NET language such as C# or Visual Basic, and the execution of the procedure or function takes place outside the database server (that is, within a separate .NET executable).

```
CREATE FUNCTION clr_interface(
    IN p1 INT,
    IN p2 UNSIGNED SMALLINT,
    IN p3 LONG VARCHAR)
RETURNS INT
EXTERNAL NAME 'CLRlib.dll::CLRproc.Run( int, ushort, string )'
LANGUAGE CLR;
```

EXTERNAL NAME *perl-call* clause To call a Perl function in an external environment, the function interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE PERL attribute.

A Perl stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in Perl and the execution of the procedure or function takes place outside the database server (that is, within a Perl executable instance).

```
CREATE FUNCTION PerlWriteToConsole( IN str LONG VARCHAR)
RETURNS INT
EXTERNAL NAME '<file=PerlConsoleExample>
    WriteToServerConsole( $sa_perl_arg0 )'
LANGUAGE PERL;
```


EXTERNAL NAME *php-call* clause To call a PHP function in an external environment, the function interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE PHP attribute.

A PHP stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in PHP and the execution of the procedure or function takes place outside the database server (that is, within a PHP executable instance).

```
CREATE FUNCTION PHPPopulateTable()
RETURNS INT
EXTERNAL NAME '<file=ServerSidePHPExample> ServerSidePHPSub()'
LANGUAGE PHP;
```

EXTERNAL NAME *java-call* clause To call a Java method in an external environment, the function interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE JAVA attribute.

A Java-interfacing stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in Java and the execution of the procedure or function takes place outside the database server (that is, within a Java VM).

```
CREATE FUNCTION HelloDemo( IN name LONG VARCHAR )
RETURNS INT
EXTERNAL NAME 'Hello.main([Ljava/lang/String;)V'
LANGUAGE JAVA;
```

The descriptors for arguments and return values from Java methods have the following meanings:

Field type	Java data type
B	byte
C	char
D	double
F	float
I	int
J	long
L <i>class-name</i> ;	An instance of the class <i>class-name</i> . The class name must be fully qualified, and any dot in the name must be replaced by a /. For example, java/lang/String .
S	short
V	void
Z	Boolean
[Use one for each dimension of an array.

EXTERNAL NAME *js-call* clause To call a JavaScript function in an external environment, the procedure interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE JS attribute.

A JavaScript stored procedure or function behaves the same as a SQL stored procedure or function except that the code is written in JavaScript and the code execution takes place outside the database server (that is, within a Node.js executable instance).

Specify the return type of the JavaScript function at the beginning of the EXTERNAL NAME string inside angle brackets. Since JavaScript does not allow pass-by-reference for simple variables inside functions, the left bracket character ([]) can proceed the S, B, I, U, or D characters to indicate that a one element array is being passed to the JavaScript stored procedure. This syntax is provided to support INOUT and OUT parameters in stored procedures.

The following is a sample function definition.

```
CREATE FUNCTION SimpleJSDemo(  
    IN thousands INT,  
    IN hundreds INT,  
    IN tens INT,  
    IN ones INT)  
RETURNS INT  
EXTERNAL NAME '<I><file=SimpleJSExample> SimpleJSFunction(IIII)'  
LANGUAGE JS;
```

The descriptors for arguments and return values from JavaScript methods have the following meanings:

Field type	JavaScript data type
S	String
B	Boolean
I	Integer
U	Unsigned integer
D	Double

Remarks

The CREATE FUNCTION statement creates a function in the database. You can create functions for other users by specifying an owner. A function is invoked as part of a SQL expression.

When referencing a temporary table from multiple functions, a potential issue can arise if the temporary table definitions are inconsistent and statements referencing the table are cached.

Privileges

You must have the CREATE PROCEDURE and CREATE EXTERNAL REFERENCE system privileges to create external functions owned by you.

You must have the CREATE ANY PROCEDURE or CREATE ANY OBJECT system privileges, as well as the CREATE EXTERNAL REFERENCE system privilege to create external functions owned by others.

No privilege is required to create temporary functions.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** CREATE FUNCTION for an external language environment is a core feature of the ANSI/ISO SQL Standard, though some of its components supported in the software are optional ANSI/ISO SQL Language Features. A subset of these features include:
 - The SQL SECURITY clause is optional ANSI/ISO SQL Language Feature T324.
 - The ability to pass a LONG VARCHAR, LONG NVARCHAR, or LONG BINARY value to a SQL function is ANSI/ISO SQL Language Feature T041.
 - Support for LANGUAGE JAVA is optional ANSI/ISO SQL Language Feature J621.
 - The ability to create or modify a schema object within an external function, using statements such as CREATE TABLE or DROP TRIGGER, is Language Feature T653.
 - The ability to use a dynamic-SQL statement within an external function, including statements such as CONNECT, EXECUTE IMMEDIATE, PREPARE, and DESCRIBE, is Language Feature T654.
- Several clauses of the CREATE FUNCTION statement are not in the standard. These include:
 - Support for C_ESQL32, C_ESQL64, C_ODBC32, C_ODBC64, CLR, PERL, and PHP in the LANGUAGES clause are not in the standard.
 - The format of *external-call* is implementation-defined.
 - The optional DEFAULT clause for a specific routine parameter is not in the standard.
 - The optional OR REPLACE clause is not in the standard.
- **Transact-SQL** CREATE FUNCTION for an external routine is supported by Adaptive Server Enterprise. Adaptive Server Enterprise only supports LANGUAGE JAVA as the external environment (SQL Language Feature J621) for an external function.

Related Information

[The JavaScript external environment](#)

JavaScript stored procedures and functions can be called from the database in the same manner as user-defined SQL stored procedures and functions.

[External environment support](#)

Six external runtime environments are supported. These include Embedded SQL and ODBC applications written in C/C++, and applications written in Java, JavaScript, Perl, PHP, or languages such as C# and Visual Basic that are based on the Microsoft .NET Framework Common Language Runtime (CLR).

[External call interface](#)

You can call a function in an external library from a stored procedure or function.

[The ESQL and ODBC external environments](#)

External compiled native functions that use Embedded SQL or ODBC can be called from the database in the same manner as SQL stored procedures.

[The CLR external environment](#)

CLR stored procedures and functions can be called from within the database in the same manner as SQL stored procedures.

[The Java external environment](#)

Java methods can be called from the database in the same manner as SQL stored procedures.

[The PHP external environment](#)

PHP stored procedures and functions can be called from the database in the same manner as SQL stored procedures.

[The Perl external environment](#)

Perl stored procedures and functions can be called from the database in the same manner as SQL

stored procedures.

[References to temporary tables within procedures](#)

Sharing a temporary table between procedures can cause problems if the table definitions are inconsistent.

[ALTER FUNCTION statement](#)

Modifies a function.

[CALL statement](#)

Invokes a procedure.

[CREATE FUNCTION statement](#)

Creates a user-defined SQL function in the database.

[CREATE FUNCTION statement \[Web service\]](#)

Creates a web client function that makes an HTTP or SOAP over HTTP request.

[CREATE PROCEDURE statement \[External call\]](#)

Creates an interface to a native or external procedure.

[DROP FUNCTION statement](#)

Removes a function from the database.

[GRANT statement](#)

Grant system and object-level privileges to users and roles.

[SQL data types](#)

There are many SQL data types supported by the software.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

CREATE FUNCTION statement [Web service]

Creates a web client function that makes an HTTP or SOAP over HTTP request.

Syntax

```
CREATE [ OR REPLACE ] FUNCTION [ owner.]function-name ( [ parameter, ... ] )
RETURNS data-type
URL url-string
[ HEADER header-string ]
[ SOAPHEADER soap-header-string ]
[ TYPE {
  'HTTP' : { GET | POST[:MIME-type] | PUT[:MIME-type] | DELETE | HEAD | OPTIONS } ' ' |
  'SOAP' : { RPC | DOC } ' ' } ]
[ NAMESPACE namespace-string ]
[ CERTIFICATE certificate-string ]
[ CLIENTPORT clientport-string ]
[ PROXY proxy-string ]
[ SET protocol-option-string ]
```

parameter :

```
[ IN ] parameter-name data-type [ DEFAULT expression ]
```

url-string :

```
' { HTTP : | HTTPS : | HTTPS_FIPS : } // [ user : password @ ] hostname [ : port ] [ / path ] '
```

protocol-option-string

```
[ http-option-list ]
```

```
[ , soap-option-list ]
```

http-option-list :

```
HTTP(
[ CH[UNK] = { ON | OFF | AUTO } ]
[ ; VER[SION] = { 1.0 | 1.1 } ; kto = number-of-seconds ]
[ REDIR(COUNT = count, STATUS = status-list)
)
```

soap-option-list :

```
SOAP(OP[ERATION] = soap-operation-name)
```

Parameters

OR REPLACE clause Specifying CREATE OR REPLACE FUNCTION creates a new function, or replaces an existing function with the same name. This clause changes the definition of the function, but preserves existing privileges. You cannot use the OR REPLACE clause with temporary functions.

parameter-name Parameter names must conform to the rules for database identifiers. They must have a valid SQL data type, and must be prefixed by the keyword IN, signifying that the argument is an expression that provides a value to the function. However, function parameters are IN by default.

data-type The data type of the parameter. You can set the data type explicitly, or specify the %TYPE or %ROWTYPE attribute to set the data type to the data type of another object in the database. Use %TYPE to set it to the data type of a column in a table or view. Use %ROWTYPE to set the data type to a composite data type derived from a row in a table or view.

RETURNS clause Specify one of the following to define the return type for the SOAP or HTTP function:

- CHAR
- VARCHAR
- LONG VARCHAR
- TEXT
- NCHAR
- NVARCHAR
- LONG NVARCHAR
- NTEXT
- XML
- BINARY
- VARBINARY
- LONG BINARY

The value returned is the body of the HTTP response. No HTTP header information is included. If more information is required, such as status information, use a procedure instead of a function.

The data type does not affect how the HTTP response is processed.

URL clause Use the URL clause only when defining an HTTP or SOAP web services client function. The URL clause specifies the URI of the web service. The optional user name and password parameters provide a means of supplying the credentials needed for HTTP basic authentication. HTTP basic authentication base-64 encodes the user and password information and passes it in the Authentication header of the HTTP request.

Specifying HTTPS_FIPS forces the system to use the FIPS-certified libraries. If HTTPS_FIPS is specified, but no FIPS-certified libraries are present, libraries that are not FIPS-certified are used instead.

For functions of type HTTP:GET, query parameters can be specified within the URL clause in addition to being automatically generated from parameters passed to a function.

URL 'http://localhost/service?parm=1

HEADER clause When creating HTTP web service client functions, use this clause to add or modify HTTP request header entries. Only printable ASCII characters can be specified for HTTP headers, and they are case-insensitive.

SOAPHEADER clause When declaring a SOAP web service as a function, use this clause to specify one or more SOAP request header entries. A SOAP header can be declared as a static constant, or can be dynamically set using the parameter substitution mechanism (declaring IN, OUT, or INOUT parameters for hd1, hd2, and so on). A web service function can define one or more IN mode substitution parameters, but cannot define an INOUT or OUT substitution parameter.

TYPE clause The TYPE clause specifies the format used when making the web service

request. SOAP:RPC is used when SOAP is specified or no TYPE clause is included. HTTP:POST is used when HTTP is specified.

The TYPE clause allows the specification of a MIME-type for HTTP:POST and HTTP:PUT types. When HTTP:PUT is used, then a MIME-type must be specified. The *MIME-type* specification is used to set the Content-Type request header and set the mode of operation to allow only a single call parameter to populate the body of the request. Only zero or one parameter may remain when making a web service function call after parameter substitutions have been processed. Calling a web service function with a NULL value or no parameter (after substitutions) results in a request with no body and a content-length of zero. When a MIME-type is specified then the single body parameter is sent in the request as is, so the application must ensure that the content is formatted to match the MIME-type.

Some typical MIME-types include:

- text/plain
- text/html
- text/xml

When no MIME-type is specified, parameter names and values (multiple parameters are permitted) are URL encoded within the body of the HTTP request.

The keywords for the TYPE clause have the following meanings:

- **HTTP:GET** By default, this type uses the application/x-www-form-urlencoded MIME-type for encoding parameters specified in the URL.

For example, the following request is produced when a client submits a request from the URL <http://localhost/WebServiceName?arg1=param1&arg2=param2>:

```
GET /WebServiceName?arg1=param1&arg2=param2 HTTP/1.1
// <End of Request - NO BODY>
```

- **HTTP:POST** By default, this type uses the application/x-www-form-urlencoded MIME-type for encoding parameters specified in the body of a POST request. URL parameters are stored in the body.

For example, the following request is produced when a client submits a request from the URL <http://localhost/WebServiceName?arg1=param1&arg2=param2>:

```
POST /WebServiceName HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 19

arg1=param1&arg2=param2
// <End of Request>
```

- **HTTP:PUT** HTTP:PUT is similar to HTTP:POST, but the HTTP:PUT type does not have a default media type.

The following example demonstrates how to configure a general purpose client function that uploads data to a database server running the `%SQLANYSAM17%\SQLAnywhere\HTTP\put_data.sql` sample:

```
CREATE OR REPLACE FUNCTION CPUT([data] LONG VARCHAR, resnm LONG VARCHAR,
mediatype LONG VARCHAR)
  RETURNS LONG BINARY
  URL 'http://localhost/resource/!resnm'
  TYPE 'HTTP:PUT:!mediatype';
```

```
SELECT CPUT('hello world', 'hello', 'text/plain' );
```

- **HTTP:DELETE** A web service client function can be configured to delete a resource located on a server. Specifying the media type is optional.

The following example demonstrates how to configure a general purpose client function that deletes a resource from a database server running the put_data.sql sample:

```
CREATE OR REPLACE FUNCTION CDEL(resnm LONG VARCHAR, mediatype LONG VARCHAR)
  RETURNS LONG BINARY
  URL 'http://localhost/resource/!resnm'
  TYPE 'HTTP:DELETE:~mediatype';
```

```
SELECT CDEL('hello', 'text/plain' );
```

- **HTTP:HEAD** The HEAD method is identical to a GET method but the server does not return a body. A media type can be specified.

```
CREATE OR REPLACE FUNCTION CHEAD(resnm LONG VARCHAR)
  RETURNS LONG BINARY
  URL 'http://localhost/resource/!resnm'
  TYPE 'HTTP:HEAD';
```

```
SELECT CHEAD( 'hello' );
```

- **HTTP:OPTIONS** The OPTIONS method is identical to a GET method but the server does not return a body. A media type can be specified. This method allows cross-origin resource sharing (CORS).
- **SOAP:RPC** This type sets the Content-Type to 'text/xml'. SOAP operations and parameters are encapsulated in SOAP envelope XML documents.
- **SOAP:DOC** This type sets the Content-Type to 'text/xml'. It is similar to the SOAP:RPC type but allows you to send richer data types. SOAP operations and parameters are encapsulated in SOAP envelope XML documents.

Specifying a MIME-type for the TYPE clause automatically sets the Content-Type header to that MIME-type.

NAMESPACE clause Applies to SOAP client functions only. This clause identifies the method namespace usually required for both SOAP:RPC and SOAP:DOC requests. The SOAP server handling the request uses this namespace to interpret the names of the entities in the SOAP request message body. The namespace can be obtained from the WSDL (Web Services Description Language) of the SOAP service available from the web service server. The default value is the function's URL, up to but not including, the optional path component.

You can specify a variable name for *namespace-string*. If the variable is NULL, the namespace property is ignored.

CERTIFICATE clause To make a secure (HTTPS) request, a client must have access to the certificate used to sign the HTTP server's certificate (or any certificate higher in the signing chain). The necessary information is specified in a string of semicolon-separated keyword=value pairs. The following keywords are available:

Keyword	Abbreviation	Description
file		The file name of the certificate or specify * to use a certificate from the operating system certificate store. Cannot be specified if either the certificate or certificate_name keyword is specified.
certificate	cert	The certificate itself. Cannot be specified if either the file or certificate_name keyword is specified.
certificate_name	cert_name	The name of a certificate stored in the database. Cannot be specified if either the file or certificate keyword is specified.
company	co	The company specified in the certificate.
unit		The company unit specified in the certificate.
name		The common name specified in the certificate.
skip_certificate_name_check		Specify ON to prevent checking the database server certificate. <div>Note Setting this option to ON is not recommended because this setting prevents the database server from fully authenticating the HTTP server.</div>

Certificates are required only for requests that are either directed to an HTTPS server, or can be redirected from a non-secure to a secure server. Only PEM formatted certificates are supported.

CLIENTPORT clause Identifies the port number on which the HTTP client function communicates using TCP/IP. It is provided for and recommended only for connections through

firewalls that filter "outgoing" TCP/IP connections. You can specify a single port number, ranges of port numbers, or a combination of both; for example, `CLIENTPORT '85,90-97'`.

PROXY clause

Specifies the URI of a proxy server. For use when the client must access the network through a proxy. The *proxy-string* is usually an HTTP or HTTPS url-string. This is site specific information that you usually need to obtain from your network administrator. This clause indicates that the function is to connect to the proxy server and send the request to the web service through it. For an example, the following PROXY clause sets the proxy server to `proxy.example.com`:

```
PROXY http://proxy.example.com
```

SET clause Specifies protocol-specific behavior options for HTTP and SOAP. The following list describes the supported SET options. CHUNK and VERSION apply to the HTTP protocol, and OPERATION applies to the SOAP protocol. Parameter substitution is supported for this clause.

- o **'HTTP(CH[UNK]=option)'** (HTTP or SOAP) This option allows you to specify whether to use chunking. Chunking allows HTTP messages to be broken up into several parts. Possible values are ON (always chunk), OFF (never chunk), and AUTO (chunk only if the contents, excluding auto-generated markup, exceeds 8196 bytes). For example, the following SET clause enables chunking:

```
SET 'HTTP(CHUNK=ON)'
```

If the CHUNK option is not specified, the default behavior is AUTO. If a chunked request fails in AUTO mode with a status of 505 `HTTP Version Not Supported`, or with 501 `Not Implemented`, or with 411 `Length Required`, the client retries the request without chunked transfer-coding.

Set the CHUNK option to OFF (never chunk) if the HTTP server does not support chunked transfer-coded requests.

Since CHUNK mode is a transfer encoding supported starting in HTTP version 1.1, setting CHUNK to ON requires that the version (VER) be set to 1.1, or not be set at all, in which case 1.1 is used as the default version.

- o **'HTTP(VER[SION]=ver;kto=number-of-seconds)'** (HTTP or SOAP) This option allows you to specify the version of HTTP protocol that is used for the format of the HTTP message. For example, the following SET clause sets the HTTP version to 1.1:

```
SET 'HTTP(VERSION=1.1)'
```

Possible values are 1.0 and 1.1. If VERSION is not specified:

- o if CHUNK is set to ON, 1.1 is used as the HTTP version
- o if CHUNK is set to OFF, 1.0 is used as the HTTP version
- o if CHUNK is set to AUTO, either 1.0 or 1.1 is used, depending on whether the client is sending in CHUNK mode
- o **kto=number-of-seconds** Specifies the keep-alive timeout criteria (kto), permitting a web client function to instantiate and cache a keep-alive HTTP/HTTPS connection for a period of time. To cache an HTTP keep-alive connection, the HTTP version must be set to 1.1 and kto set to a non-zero value. kto may be useful, for HTTPS connections particularly, if you notice a significant performance difference between HTTP and HTTPS connections. A database connection can only cache a

single keep-alive HTTP connection. Subsequent calls to a web client function using the same URI reuse the keep-alive connection. Therefore, the executing web client call must have a URI whose scheme, destination host and port match that of the cached URI, and the HEADER clause must not specify Connection: close. When `kto` is not specified, or is set to zero, HTTP/HTTPS connections are not cached.

- **'REDIR(COUNT=*count*, STATUS=*status-list*)'** (HTTP or SOAP) The HTTP response status codes such as 302 Found and 303 See Other are used to redirect web applications to a new URI, particularly after an HTTP POST has been performed. For example, a client request could be:

```
GET /people/alice HTTP/1.1
Host: www.example.com
Accept: text/html, application/xhtml+xml
Accept-Language: en, de
```

The web server response could be:

```
HTTP/1.1 302 Found
Location: http://www.example.com/people/alice.en.html
```

In response, the client would send another HTTP request to the new URI. The REDIR option allows you to control the maximum number of redirections allowed and which HTTP response status codes to automatically redirect.

For example, `SET 'REDIR(count=3, status=301,307)'` allows a maximum limit of 3 re-directions and permits redirection for 301 and 307 statuses. If one of the other redirection status codes such as 302 or 303 is received, an error is issued (SQLCODE -983).

The default redirection limit *count* is 5. By default, an HTTP client function will automatically redirect in response to all HTTP redirection status codes (301, 302, 303, 307). To disallow all redirection status codes, use `SET REDIR(COUNT=0)`. In this mode, a redirection response does not result in an error (SQLCODE -983). Instead, a result set is returned with the HTTP status and response headers. This permits a caller to conditionally reissue the request based on the URI contained in the Location header.

A web service function specifying a POST HTTP method which receives a 303 See Other status issues a redirect request using the GET HTTP method.

The Location header can contain either an absolute path or a relative path. The HTTP client function handles either. The header can also include query parameters and these are forwarded to the redirected location. For example, if the header contained parameters such as the following, the subsequent GET or a POST includes these parameters.

```
Location: alternate_service?a=1&b=2
```

In the above example, the query parameters are `a=1&b=2`.

- **'SOAP(OP[ERATION]=*soap-operation-name*)'** (SOAP only) This option allows you to specify the name of the SOAP operation, if it is different from the name of the function you are creating. The value of OPERATION is analogous to the name of a remote function call. For example, if you wanted to create a function called `accounts_login` that calls a SOAP operation called `login`, you would specify something like the following:

```
CREATE FUNCTION accounts_login( name LONG VARCHAR, pwd LONG VARCHAR )  
RETURNS LONG BINARY  
SET 'SOAP( OPERATION=login )'
```

If the OPERATION option is not specified, the name of the SOAP operation must match the name of the function you are creating.

The following statement shows how several *protocol-option-string* settings are combined in the same SET clause:

```
CREATE FUNCTION accounts_login( name LONG VARCHAR, pwd LONG VARCHAR )  
RETURNS LONG BINARY  
SET 'HTTP( CHUNK=ON; VERSION=1.1 ), SOAP( OPERATION=login )'  
...
```

Remarks

The CREATE FUNCTION statement creates a web services function in the database. A function can be created for another user by specifying an owner name.

When functions are executed, not all parameters need to be specified. If a DEFAULT value is provided in the CREATE FUNCTION statement, missing parameters are assigned the default values. If an argument is not provided by the caller and no default is set, an error is given.

Parameter values are passed as part of the request. The syntax used depends on the type of request. For HTTP:GET, the parameters are passed as part of the URL; for HTTP:POST requests, the values are placed in the body of the request. Parameters to SOAP requests are always bundled in the request body.

Note For *required* parameters that accept variable names, an error is returned if one of the following conditions is true:

- The variable does not exist
- The contents of the variable are NULL
- The variable exceeds the length allowed by the parameter
- The data type of the variable does not match that required by the parameter

Privileges

You must have the CREATE PROCEDURE system privilege to create functions owned by you.

You must have the CREATE ANY PROCEDURE or CREATE ANY OBJECT system privilege to create functions owned by others.

You must have the CREATE EXTERNAL REFERENCE system privilege to create an external function.

No privilege is required to create temporary functions.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

- **Transact-SQL** Not supported by Adaptive Server Enterprise.

Example

1. The following statement creates a function named `cli_test1` that returns images from the `get_picture` service running on localhost:

```
CREATE FUNCTION cli_test1( image LONG VARCHAR )
RETURNS LONG BINARY
URL 'http://localhost/get_picture'
TYPE 'HTTP:GET';
```

2. The following statement issues an HTTP request with the URL `http://localhost/get_picture?image=widget`:

```
SELECT cli_test1( 'widget' );
```

3. The following statement uses a substitution parameter to allow the request URL to be passed as an input parameter. The secure HTTPS request uses a certificate stored in the database. The SET clause is used to turn off CHUNK mode transfer-encoding.

```
CREATE CERTIFICATE client_cert
FROM FILE 'C:\\Users\\Public\\Documents\\SQL Anywhere
          17\\Samples\\Certificates\\rsaroot.crt';

CREATE FUNCTION cli_test2( image LONG VARCHAR, myurl LONG VARCHAR )
RETURNS LONG BINARY
URL '!myurl'
CERTIFICATE 'certificate_name=client_cert'
TYPE 'HTTPS:GET'
SET 'HTTP(CH=OFF)'
HEADER 'ASA-ID';
```

4. The following statement issues an HTTP request with the URL `http://localhost/get_picture?image=widget`:

```
CREATE VARIABLE a_binary LONG BINARY;
SET a_binary = cli_test2( 'widget', 'https://localhost/get_picture' )
SELECT a_binary;
```

5. The following example creates a function using a variable in the NAMESPACE clause
 - a. The following statements create a variable for a NAMESPACE clause:

```
CREATE VARIABLE @ns LONG VARCHAR
SET @ns = 'http://wsdl.domain.com/';
```

- b. The following statement creates a function named `FtoC` that uses a variable in the NAMESPACE clause:

```
CREATE FUNCTION FtoC ( IN temperature LONG VARCHAR )  
RETURNS LONG VARCHAR  
URL 'http://localhost:8082/FtoCService'  
TYPE 'SOAP:DOC'  
NAMESPACE @ns;
```

Related Information

[HTTP and SOAP request structures](#)

All parameters to a function or procedure, unless used during parameter substitution, are passed as part of the web service request. The format in which they are passed depends on the type of the web service request.

[HTTP request header management](#)

HTTP request headers can be added, changed, or removed with the HEADER clause of the CREATE PROCEDURE and CREATE FUNCTION statements.

[Variables supplied to web services](#)

Variables can be supplied to a web service in various ways depending on the web service type.

[The database server as an HTTP web server](#)

The database server contains a built-in HTTP web server that allows you to provide online web services from a database.

[Web client application development](#)

SQL Anywhere databases can act as web client applications to access web services hosted on a SQL Anywhere web server or on third party web servers.

[%TYPE and %ROWTYPE attributes](#)

In addition to explicitly setting the data type for an object, you can also set the data type by specifying the %TYPE and %ROWTYPE attributes.

[Tutorial: Using a database server to access a SOAP/DISH service](#)

This tutorial illustrates how to create a SOAP server that converts a web client-supplied Fahrenheit temperature value to Celsius.

[Tutorial: Create a web server and access it from a web client](#)

This tutorial illustrates how to create a web server and then send requests to it from a web client database server.

[SQL data types](#)

There are many SQL data types supported by the software.

[ALTER FUNCTION statement](#)

Modifies a function.

[CREATE FUNCTION statement](#)

Creates a user-defined SQL function in the database.

[CREATE FUNCTION statement \[External call\]](#)

Creates an interface to a native or external function.

[CREATE PROCEDURE statement](#)

Creates a user-defined SQL procedure in the database.

[CREATE PROCEDURE statement \[Web service\]](#)

Creates a user-defined web client procedure that makes HTTP or SOAP requests to an HTTP server.

[DROP FUNCTION statement](#)

Removes a function from the database.

[RETURN statement](#)

Exits from a function, procedure, or batch unconditionally, optionally providing a return value.

[remote_idle_timeout option](#)

Controls how many seconds of inactivity web service client procedures and functions tolerate.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

CREATE FUNCTION statement

Creates a user-defined SQL function in the database.

Syntax

```
CREATE [ OR REPLACE | TEMPORARY ] FUNCTION [ owner.]function-name
( [ parameter, ... ] )
RETURNS data-type
[ SQL SECURITY { INVOKER | DEFINER } ]
[ ON EXCEPTION RESUME ]
[ [ NOT ] DETERMINISTIC ]
compound-statement | AS tsq-compound-statement | AT location-string

parameter :
  [ IN ] parameter-name data-type [ DEFAULT expression ]

tsq-compound-statement :
  sql-statement
  sql-statement
  ...
```

Parameters

OR REPLACE clause Specifying CREATE OR REPLACE FUNCTION creates a new function, or replaces an existing function with the same name. When a function is replaced, the definition of the function is changed but the existing privileges are preserved.

You cannot use the OR REPLACE clause with temporary functions.

TEMPORARY keyword Specifying CREATE TEMPORARY FUNCTION means that the function is visible only by the connection that created it, and that it is automatically dropped when the connection is dropped. Temporary functions can also be explicitly dropped. You cannot perform ALTER, GRANT, or REVOKE on them, and, unlike other functions, temporary functions are not recorded in the catalog or transaction log.

Temporary functions execute with the privileges of their creator (current user) or specified owner. You can specify an owner for a temporary function when:

- the temporary function is created within a permanent stored procedure
- the owner of the temporary function and permanent stored procedure is the same

To drop the owner of a temporary function, you must drop the temporary function first.

Temporary functions can be created and dropped when connected to a read-only database.

You cannot use the OR REPLACE clause with temporary functions.

parameter-name Parameter names must conform to the rules for database identifiers. They must have a valid SQL data type, and must be prefixed by the keyword IN, signifying that the argument is an expression that provides a value to the function. However, function parameters are IN by default.

data-type The data type of the parameter. Set the data type explicitly, or specify the %TYPE or %ROWTYPE attribute to set the data type to the data type of another object in the database.

Use %TYPE to set it to the data type of a column in a table or view. Use %ROWTYPE to set the data type to a composite data type derived from a row in a table or view.

RETURNS clause Use the RETURNS clause to specify the data type for the result of the function.

If a RETURNS clause is specified, it must be the first clause of the statement.

SQL SECURITY clause The SQL SECURITY clause defines whether the function is executed as the INVOKER (the user who is calling the function), or as the DEFINER (the user who owns the function). The default is DEFINER.

ON EXCEPTION RESUME clause Use Transact-SQL-like error handling.

[NOT] DETERMINISTIC clause Use this clause to indicate whether functions are deterministic or non-deterministic. If this clause is omitted, then the deterministic behavior of the function is unspecified (the default).

If a function is declared as DETERMINISTIC, it should return the same value every time it is invoked with the same set of parameters.

If a function is declared as NOT DETERMINISTIC, then it is not guaranteed to return the same value for the same set of parameters. A function declared as NOT DETERMINISTIC is re-evaluated each time it is called in a query. This clause must be used when it is known that the function result for a given set of parameters can vary.

Also, functions that have side effects such as modifying the underlying data should be declared as NOT DETERMINISTIC. For example, here is an example of a function that generates primary key values and is used in an INSERT...SELECT statement; it is declared NOT DETERMINISTIC. The tables that are referenced are fictitious.

```
CREATE FUNCTION keygen( increment INTEGER )
RETURNS INTEGER
NOT DETERMINISTIC
BEGIN
    DECLARE keyval INTEGER;
    UPDATE counter SET x = x + increment;
    SELECT counter.x INTO keyval FROM counter;
    RETURN keyval
END
INSERT INTO new_table
SELECT keygen(1), ...
FROM old_table;
```

Functions can be declared as DETERMINISTIC if they always return the same value for given input parameters.

compound-statement A set of SQL statements bracketed by BEGIN and END, and separated by semicolons.

AS clause *tsql-compound-statement* is a batch of Transact-SQL statements.

AT clause Create a proxy function on the current database for a remote function specified by *location-string*. The AT clause supports the semicolon (;) as a field delimiter in *location-string*. If no semicolon is present, a period is the field delimiter. The use of semicolons allows file names and extensions to be used in the database and owner fields.

The string in the AT clause can also contain local or global variable names enclosed in braces

(*{variable-name}*). The SQL variable name must be of type CHAR, VARCHAR, or LONG VARCHAR. For example, an AT clause that contains '*bostonase.master.dbo.{@myfunction}*' indicates that *@myfunction* is a SQL variable and that the current contents of the *@myfunction* variable should be substituted when the remote procedure is used.

A proxy function can return any data type except DECIMAL, NUMERIC, LONG VARCHAR, LONG NVARCHAR, LONG BINARY, XML, or any spatial data type.

Remarks

The CREATE FUNCTION statement creates a function in the database. A function can be created for another user by specifying an owner name. Subject to privileges, a function can be used in exactly the same way as other non-aggregate functions.

When functions are executed, not all parameters need to be specified. If a DEFAULT value is provided in the CREATE FUNCTION statement, missing parameters are assigned the default values. If an argument is not provided by the caller and no default is set, an error is given.

When SQL SECURITY INVOKER is specified, more memory is used because annotation must be done for each user that calls the procedure. Also, when SQL SECURITY INVOKER is specified, name resolution is done as the invoker. Therefore, make sure to qualify all object names (tables, procedures, and so on) with their appropriate owner.

All functions are treated as deterministic unless they are declared NOT DETERMINISTIC. Deterministic functions return a consistent result for the same parameters, and are free of side effects. That is, the database server assumes that two successive calls to the same function with the same parameters returns the same result, and does not have any unwanted side effects on the query's semantics.

If a function returns a result set, it cannot also set output parameters or return a return value.

Privileges

You must have the CREATE PROCEDURE system privilege to create functions owned by you.

You must have the CREATE ANY PROCEDURE or CREATE ANY OBJECT system privilege to create functions owned by others.

You must also have the CREATE EXTERNAL REFERENCE system privilege to create an external function.

No privilege is required to create temporary functions.

To replace an existing function, you must be the owner of the function, or have one of the following:

- CREATE ANY FUNCTION and DROP ANY FUNCTION system privileges.
- CREATE ANY OBJECT and DROP ANY OBJECT system privileges.
- ALTER ANY OBJECT or ALTER ANY FUNCTION system privileges.

Side effects

Automatic commit, even for temporary functions.

Standards

- **ANSI/ISO SQL Standard** CREATE FUNCTION is a core feature of ANSI/ISO SQL Standard, though some of its components supported in the software are optional SQL Language Features. A subset of these features includes:

- The SQL SECURITY clause is optional Language Feature T324.
- The ability to pass a LONG VARCHAR, LONG NVARCHAR, or LONG BINARY value to a SQL function is Language Feature T041.
- The ability to create or modify a schema object within a SQL function, using statements such as CREATE TABLE or DROP TRIGGER, is Language Feature T651.
- The ability to use a dynamic-SQL statement within a SQL function, including statements such as EXECUTE IMMEDIATE, PREPARE, and DESCRIBE, is Language Feature T652.

Several clauses of the CREATE FUNCTION statement are not in the standard. These include:

- The TEMPORARY clause.
 - The ON EXCEPTION RESUME clause.
 - The optional DEFAULT clause for a specific routine parameter.
 - The specification of a Transact-SQL function using the AS clause.
 - The optional OR REPLACE clause.
- **Transact-SQL** CREATE FUNCTION is supported by Adaptive Server Enterprise. Adaptive Server Enterprise does not support the optional IN keyword for function parameters.

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 example replaces the fullname function created in the first example. After replacing the function, the local variable `name` is removed:

```
CREATE OR REPLACE FUNCTION fullname(
    firstname CHAR(30),
    lastname CHAR(30) )
RETURNS CHAR(61)
BEGIN
    RETURN ( firstname || ' ' || lastname );
END;
```

The following examples illustrate the use of the fullname function.

Return a full name from two supplied strings:

```
SELECT fullname ( 'joe', 'smith' );
```

fullname('joe', 'smith')

fullname('joe', 'smith')
joe smith

List the names of all employees:

```
SELECT fullname ( GivenName, Surname )
FROM GROUP0.Employees;
```

fullname (GivenName, Surname)
Fran Whitney
Matthew Cobb
Philip Chin
Julie Jordan
...

The following function uses Transact-SQL syntax:

```
CREATE FUNCTION DoubleIt( @Input INT )
RETURNS INT
AS
BEGIN
    DECLARE @Result INT
    SELECT @Result = @Input * 2
    RETURN @Result
END;
```

The statement `SELECT DoubleIt(5);` returns a value of 10.

The following example creates a function called fullname and sets the data types of the firstname and lastname parameters to the data types of the Surname and Givenname column of the Employees table by using a %TYPE attribute:

```
CREATE OR REPLACE FUNCTION fullname(
    firstname Employees.Surname%TYPE,
    lastname Employees.GivenName%TYPE )
RETURNS LONG VARCHAR
BEGIN
    RETURN ( firstname || ' ' || lastname );
END;

SELECT fullname ( Surname, GivenName )FROM GROUP0.Employees;
```

Related Information

[Stored procedures, triggers, batches, and user-defined functions](#)

Procedures and triggers store procedural SQL statements in a database.

[Transact-SQL batches](#)

In Transact-SQL, a batch is a set of SQL statements submitted together and executed as a group.

[%TYPE and %ROWTYPE attributes](#)

In addition to explicitly setting the data type for an object, you can also set the data type by specifying the %TYPE and %ROWTYPE attributes.

[Creating a user-defined function](#)

Create a user-defined function using SQL Central.

[ALTER FUNCTION statement](#)

Modifies a function.

[CREATE FUNCTION statement \[External call\]](#)

Creates an interface to a native or external function.

[CREATE FUNCTION statement \[Web service\]](#)

Creates a web client function that makes an HTTP or SOAP over HTTP request.

[BEGIN statement](#)

Specifies a compound statement.

[CREATE PROCEDURE statement](#)

Creates a user-defined SQL procedure in the database.

[DROP FUNCTION statement](#)

Removes a function from the database.

[RETURN statement](#)

Exits from a function, procedure, or batch unconditionally, optionally providing a return value.

[SQL data types](#)

There are many SQL data types supported by the software.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

CREATE INDEX statement

Creates an index on a specified table or materialized view.

Syntax

- **Creating an index on a table**

```
CREATE [ VIRTUAL ] [ UNIQUE ] [ CLUSTERED ] INDEX [ IF NOT EXISTS ] index-name
ON [ owner. ] table-name
    ( column-name [ ASC | DESC ], ...
      | function-name ( argument, [ ... ] ) AS column-name )
| [ WITH NULLS [ NOT ] DISTINCT ]
| [ { IN | ON } dbspace-name ]
| [ FOR OLAP WORKLOAD ]
```

- **Creating an index on a materialized view**

```
CREATE [ VIRTUAL ] [ UNIQUE ] [ CLUSTERED ] INDEX [ IF NOT EXISTS ]
index-name
ON [ owner. ] materialized-view-name
    ( column-name [ ASC | DESC ], ... )
| [ WITH NULLS NOT DISTINCT ]
| [ { IN | ON } dbspace-name ]
| [ FOR OLAP WORKLOAD ]
```

Parameters

VIRTUAL clause The VIRTUAL keyword is primarily for use by the Index Consultant. A virtual index mimics the properties of a real physical index during the evaluation of execution plans by the Index Consultant and when the PLAN function is used. You can use virtual indexes together with the PLAN function to explore the performance impact of an index, without the often time-consuming and resource-consuming effects of creating a real index.

Virtual indexes are not visible to other connections, and are dropped when the connection is closed. Virtual indexes are not used when evaluating plans for the actual execution of queries, and so do not interfere with performance.

Virtual indexes have a limit of four columns.

UNIQUE clause The UNIQUE attribute ensures that there will not be two rows in the table or materialized view with identical values in all the columns in the index. If you specify UNIQUE, but do not specify WITH NULLS NOT DISTINCT, each index key must be unique or contain a NULL in at least one column. For example, two entries ('a', NULL) and ('a', NULL) are each considered unique.

If you specify UNIQUE...WITH NULLS NOT DISTINCT, then the index key must be unique regardless of the NULL values. For example, two entries ('a', NULL) and ('a', NULL) are considered equal, not unique.

There is a difference between a unique constraint and a unique index. Columns of a unique index are allowed to be NULL, while columns in a unique constraint are not. A foreign key can reference either a primary key or a unique constraint, but not a unique index, because it can

include multiple instances of NULL.

It is recommended that you do not use approximate data types such as FLOAT and DOUBLE for primary keys or for columns in unique constraints. Approximate numeric data types are subject to rounding errors after arithmetic operations.

Spatial columns cannot be included in a unique index.

CLUSTERED clause The CLUSTERED attribute causes rows to be stored in an approximate key order corresponding to the index. While the database server makes an attempt to preserve key order, total clustering is not guaranteed.

If a clustered index exists, the LOAD TABLE statement inserts rows in the order of the index key, and the INSERT statement attempts to put new rows on the same page as the one containing adjacent rows, as defined by the key order.

IF NOT EXISTS clause When the IF NOT EXISTS attribute is specified and the named index already exists, no changes are made and an error is not returned.

ASC | DESC clause Columns are sorted in ascending (increasing) order unless descending (DESC) is explicitly specified. An index is used for both an ascending and a descending ORDER BY, whether the index was ascending or descending. However, if an ORDER BY is performed with mixed ascending and descending attributes, an index is used only if the index was created with the same ascending and descending attributes.

function-name The function-name clause creates an index on a function. This clause cannot be used on declared temporary tables or materialized views.

This form of the CREATE INDEX statement is a convenience method that carries out the following operations:

1. Adds a computed column named *column-name* to the table. The column is defined with a COMPUTE clause that is the specified function, along with any specified arguments. See the COMPUTE clause of the CREATE TABLE statement for restrictions on the type of function that can be specified. The data type of the column is based on the result type of the function.
2. Populates the computed column for the existing rows in the table.
3. Creates an index on the column.

Dropping the index does not cause the associated computed column to be dropped.

IN | ON clause By default, the index is placed in the same database file as its table or materialized view. You can place the index in a separate database file by specifying a dbspace name in which to put the index. This feature is useful mainly for large databases to circumvent file size limitations, or for performance improvements that might be achieved by using multiple disk devices.

If the new index can share the physical index with an existing logical index, the IN clause is ignored.

WITH NULLS NOT DISTINCT clause This clause can only be specified if you are declaring the index to be UNIQUE and allows you to specify that NULLs in index keys are not unique. For more information, see the UNIQUE clause.

FOR OLAP WORKLOAD clause When you specify FOR OLAP WORKLOAD, the database server performs certain optimizations and gathers statistics on the key to help improve performance for OLAP workloads. Performance improvements are most noticeable when the optimization_workload is set to OLAP.

Remarks

Indexes can improve database performance. The database server uses physical and logical indexes. A physical index is the actual indexing structure as it is stored on disk. A logical index is a reference to a physical index. If you create an index that is identical in its physical attributes to an existing index, the database server creates a logical index that shares the existing physical index. In general, indexes created by users are considered logical indexes. The database server creates physical indexes as required to implement logical indexes, and can share the same physical index among several logical indexes.

The CREATE INDEX statement creates a sorted index on the specified columns of the named table or materialized view. Indexes are automatically used to improve the performance of queries issued to the database, and to sort queries with an ORDER BY clause. Once an index is created, it is never referenced in a SQL statement again except to validate it (VALIDATE INDEX), alter it (ALTER INDEX), delete it (DROP INDEX), or in a hint to the optimizer.

- **Index ownership** There is no way of specifying the index owner in the CREATE INDEX statement. Indexes are always owned by the owner of the table or materialized view.
- **Indexes on views** You can create indexes on materialized views, but not on regular views.
- **Index name space** The name of each index must be unique for a given table or materialized view.
- **Exclusive use** CREATE INDEX is prevented whenever the statement affects a table or materialized view currently being used by another connection.
- **Automatically created indexes** The database server automatically creates indexes for primary key, foreign key, and unique constraints. These automatically created indexes are held in the same database file as the table.

This statement cannot be executed when there are cursors opened with the WITH HOLD clause that use either statement or transaction snapshots.

Privileges

To create an index on a table, you must be the owner of the table, or have one of the following privileges:

- REFERENCES privilege on the table
- CREATE ANY INDEX system privilege
- CREATE ANY OBJECT system privilege

To create an index on a materialized view, you must be the owner of the materialized view, or have one of the following privileges:

- CREATE ANY INDEX system privilege
- CREATE ANY OBJECT system privilege

Side effects

Automatic commit in most cases. If the auto_commit_on_create_local_temp_index option is set to Off, there is no commit before creating an index on a local temporary table. Creating an index on a function (an implicit computed column) causes a checkpoint.

Column statistics are updated (or created if they do not exist).

The CREATE INDEX statement only applies an exclusive lock to the table for a limited time at the beginning and at the end of its execution. In the middle of the CREATE INDEX statement's execution, a shared lock is applied to the table. Statements that require exclusive access to the

table for the entirety of their execution cannot be executed concurrently with the CREATE INDEX statement even though its exclusive lock is only temporary. However, statements that only require shared access can be executed on the table, but only in the middle of the CREATE INDEX statement's execution.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

Create a two-column index on the Employees table.

```
CREATE INDEX employee_name_index
ON GROUPO.Employees
( Surname, GivenName );
```

Create an index on the SalesOrderItems table for the ProductID column.

```
CREATE INDEX item_prod
ON GROUPO.SalesOrderItems
( ProductID );
```

Use the SORTKEY function to create an index on the Description column of the Products table, sorted according to a Russian collation. As a side effect, the statement adds a computed column desc_ru to the table. For this example to succeed, you must also have SELECT privilege on the Products table.

```
CREATE INDEX ix_desc_ru
ON GROUPO.Products (
  SORTKEY( Description, 'rusdict' )
  AS desc_ru );
```

Related Information

[Advanced: Logical and physical indexes](#)

The software supports logical and physical indexes.

[Indexes](#)

An **index** provides an ordering on the rows in a column or the columns of a table.

[Optimize indexes to improve performance](#)

Obtain recommendations on the best set of indexes for your database by running the index analyzer tool, the Index Consultant.

[Snapshot isolation](#)

Snapshot isolation is designed to improve concurrency and consistency by maintaining different versions of data.

[Clustered indexes](#)

You can improve the performance of a large index scan by declaring that the index is **clustered**.

[Computed columns](#)

A computed column is an expression that can refer to the values of other columns, called **dependent columns**, in the same row.

[OLAP support](#)

On-Line Analytical Processing (OLAP) offers the ability to perform complex data analysis within a single SQL statement, increasing the value of the results, while improving performance by decreasing the amount of querying on the database.

[Physical limitations on size and number of databases](#)

This topic lists the physical limitations on size and number of objects in a SQL Anywhere database. Typically, the memory, CPU, and disk drive of the computer are the most limiting factors.

[Creating an index](#)

Create indexes on base tables, temporary tables, and materialized views.

[Obtaining index recommendations for queries \(Interactive SQL\)](#)

Access Index Consultant recommendations for a query in Interactive SQL.

[auto_commit_on_create_local_temp_index option](#)

Controls whether the database server performs a COMMIT before an index is created on a local temporary table.

[DROP INDEX statement](#)

Removes an index from the database.

[CREATE STATISTICS statement](#)

Recreates the column statistics used by the optimizer, and stores them in the ISYSCOLSTAT system table.

[optimization_workload option](#)

Determines whether query processing is optimized towards a workload that is a mix of updates and reads or a workload that is predominantly read-based (OLAP).

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

SQL Anywhere 17 » SQL Anywhere Server - SQL Reference » SQL statements
 » Alphabetical list of SQL statements

CREATE LDAP SERVER statement

Creates an LDAP server configuration object.

Syntax


```
CREATE LDAP SERVER ldapua-server-name
[ ldapua-server-attrs ... ]
[ WITH ACTIVATE ]

ldapua-server-attrs :
SEARCH DN search-dn-attributes ...
| AUTHENTICATION URL { 'url-string' | NULL }
| CONNECTION TIMEOUT timeout-value
| CONNECTION RETRIES retry-value
| TLS { ON | OFF }

search-dn-attributes :
URL { 'url-string' | NULL }
| ACCESS ACCOUNT { 'dn-string' | NULL }
| IDENTIFIED BY ( 'password' | NULL )
| IDENTIFIED BY ENCRYPTED { encrypted-password | NULL }
```

Parameters

SEARCH DN clause There is no default value for any parameter in the SEARCH DN clause.

- **URL** Use this clause to specify to identify the host (by name or by IP address), port number, and search to be performed to do the lookup of the **LDAP Distinguished Name (DN)** for a given user ID. *url-string* is validated for correct LDAP URL syntax before it is stored in ISYSLDAPSERVER. The maximum size for this string is 1024 bytes.
 The format of *url-string* must comply with the LDAP URL standard. See [The LDAP Standard Specification](#) .
- **ACCESS ACCOUNT** Use this clause to specify the DN used by the database server to connect to the LDAP server. This is not a SQL Anywhere user, but a user created in the LDAP server specifically for logging in to the LDAP server. This user must have permissions within the LDAP server to search for DN's by user ID in the locations specified in the SEARCH DN URL clause. The maximum size for this string is 1024 bytes.
- **IDENTIFIED BY** Use this clause to specify the password associated with the user identified by ACCESS ACCOUNT. The maximum size is 255 bytes and cannot be set to NULL.
- **IDENTIFIED BY ENCRYPTED** Use this clause to specify the password associated with the user identified by ACCESS ACCOUNT, provided in encrypted form, and is a binary value stored somewhere on disk. The maximum size of the binary is 289 bytes, and cannot be set to NULL. IDENTIFIED BY ENCRYPTED allows the password to be retrieved and used, without it becoming known.

AUTHENTICATION URL clause Use this clause to specify the *url-string* that identifies the host by name or IP address, and the port number of the LDAP server to use to authenticate a

user. The DN of the user obtained from a prior DN search and the user password are used to bind a new connection to the authentication URL. A successful connection to the LDAP server is considered proof of the identity of the connecting user. There is no default value for this parameter. For size limits to this string, see the SYSLDAPSERVER system view.

CONNECTION TIMEOUT clause Use this clause to specify the connection timeout, in milliseconds, to the LDAP server, both for searches for the DN and for authentication. The default value is 10 seconds.

CONNECTION RETRIES clause Use this clause to specify the number of retries for connections to the LDAP server, both for searches for the DN and for authentication. The valid range of values is 1-60. The default is 3.

TLS clause Use this clause to specify the use of the TLS protocol on connections to the LDAP server, both for the DN searches and for authentication. The valid values are ON or OFF. The default is OFF. Use the Secure LDAP protocol by using `ldaps://` to begin the URL instead of `ldap://`. The TLS option must be set to OFF when using Secure LDAP.

WITH ACTIVATE clause Use this clause to activate the LDAP server for immediate use. This clause permits the definition and activation of LDAP User Authentication in one statement, changing the state of the new LDAP server to READY.

Remarks

If you use this statement in a procedure, do not specify the password (IDENTIFIED BY clause) as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

Privileges

You must have the MANAGE ANY LDAP SERVER system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

This example sets search parameters, authentication URL, 3 second timeout, and activates the LDAP server so it can begin authenticating users. A connection is made to the LDAP server without TLS or SECURE LDAP protocols. In addition to the privileges required to execute the CREATE LDAP SERVER statement, you must also have the SET ANY SECURITY system privilege to set the login_mode option in the following example.

```
SET OPTION PUBLIC.login_mode = 'Standard,LDAPUA';
CREATE LDAP SERVER apps_primary
  SEARCH DN
    URL 'ldap://voyager:389/dc=MyCompany,dc=com??sub?cn=*'
    ACCESS ACCOUNT 'cn=aseadmin, cn=Users, dc=mycompany, dc=com'
    IDENTIFIED BY 'Secret99Password'
  AUTHENTICATION URL 'ldap://voyager:389/'
  CONNECTION TIMEOUT 3000
  WITH ACTIVATE;
```

This example uses the same search parameters, but specifies `ldaps://` so that a Secure LDAP connection is established with the LDAP server on host `voyager`, port 636. Only LDAP clients using the Secure LDAP protocol may connect on this port. The database security option `Trusted_certificate_file` must be set with a filename containing the certificate of the Certificate Authority (CA) that signed the certificate used by the LDAP server at `'ldaps://voyager:636'`. During the handshake with the LDAP server, the certificate presented by the LDAP server is verified by the database server to ensure that it is signed by one of the certificates listed in the file. The `ACCESS ACCOUNT` and `IDENTIFIED BY` parameters provided to the LDAP server are verified by the LDAP server as well.

```
SET OPTION PUBLIC.login_mode = 'Standard,LDAPUA';
SET OPTION PUBLIC.trusted_certificates_file = '/opt/sap/shared/trusted.txt';
CREATE LDAP SERVER secure_primary
  SEARCH DN
    URL 'ldaps://voyager:636/dc=MyCompany,dc=com??sub?cn=*'
    ACCESS ACCOUNT 'cn=aseadmin, cn=Users, dc=mycompany, dc=com'
    IDENTIFIED BY 'Secret99Password'
  AUTHENTICATION URL 'ldaps://voyager:636/'
  CONNECTION TIMEOUT 3000
  WITH ACTIVATE;
```

Related Information

[ALTER LDAP SERVER statement](#)

Alters an LDAP server configuration object.

[DROP LDAP SERVER statement](#)

Drops an LDAP server configuration object.

[VALIDATE LDAP SERVER statement](#)

Validates an LDAP server configuration object.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CREATE LOCAL TEMPORARY TABLE statement

Creates a local temporary table within a procedure that persists after the procedure completes and until it is either explicitly dropped, or until the connection terminates.

Syntax

```
CREATE LOCAL TEMPORARY TABLE [ IF NOT EXISTS ] table-name  
( { column-definition [ column-constraint ... ] | table-constraint | pctfree }, ... )  
[ ON COMMIT { DELETE | PRESERVE } ROWS | NOT TRANSACTIONAL ]
```

pctfree : **PCTFREE** *percent-free-space*

percent-free-space : *integer*

Parameters

IF NOT EXISTS clause No changes are made if the named table already exists, and an error is not returned.

ON COMMIT clause By default, the rows of a temporary table are deleted on a COMMIT. You can use the ON COMMIT clause to preserve rows on a COMMIT.

NOT TRANSACTIONAL clause The NOT TRANSACTIONAL clause provides performance improvements in some circumstances because operations on non-transactional temporary tables do not cause entries to be made in the rollback log. For example, NOT TRANSACTIONAL may be useful if procedures that use the temporary table are called repeatedly with no intervening COMMITs or ROLLBACKs.

Remarks

In a procedure, use the CREATE LOCAL TEMPORARY TABLE statement, instead of the DECLARE LOCAL TEMPORARY TABLE statement, when you want to create a table that persists after the procedure completes. Local temporary tables created using the CREATE LOCAL TEMPORARY TABLE statement remain until they are either explicitly dropped, or until the connection closes.

Tables created using CREATE LOCAL TEMPORARY TABLE do not appear in the SYSTABLE view of the system catalog.

Local temporary tables created in IF statements using CREATE LOCAL TEMPORARY TABLE also persist after the IF statement completes.

Privileges

None.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** CREATE LOCAL TEMPORARY TABLE is part of optional ANSI/ISO SQL Language Feature F531. The PCTFREE and NOT TRANSACTIONAL clauses are not in the standard. The column and constraint definitions defined by the statement may also include syntax extensions that are not in the standard. In the ANSI/ISO SQL Standard, tables created

via the CREATE LOCAL TEMPORARY TABLE statement appear in the system catalog; however, this is not the case in the software.

- **Transact-SQL** CREATE LOCAL TEMPORARY TABLE is not supported by Adaptive Server Enterprise. In SAP Adaptive Server Enterprise, one creates a temporary table using the CREATE TABLE statement with a table name that begins with the special character #.

Example

The following example creates a local temporary table called TempTab:

```
CREATE LOCAL TEMPORARY TABLE TempTab ( number INT )  
ON COMMIT PRESERVE ROWS;
```

Related Information

[Compound statements](#)

The body of a procedure or trigger is a **compound statement**.

[CREATE TABLE statement](#)

Creates a new table in the database and, optionally, creates a table on a remote server.

[DECLARE LOCAL TEMPORARY TABLE statement](#)

Declares a local temporary table.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CREATE LOGIN POLICY statement

Creates a login policy.

Syntax

CREATE LOGIN POLICY *policy-name policy-options*

policy options :

policy-option [*policy-option* ...]

policy-option :

policy-option-name = *policy-option-value*

policy-option-value :

{ **UNLIMITED** | *option-value* }

Parameters

policy-name The name of the login policy.

policy-option-name The name of the login policy option.

policy-option-value The value assigned to the login policy option. If you specify UNLIMITED, no limits are imposed.

Remarks

If you do not specify a policy option, then the corresponding root login policy option is always used. However, new policies do not inherit the `max_non_dba_connections` and `root_auto_unlock_time` policy options, these are root policy-only options.

For all unspecified settings, the new policy does not make static copies from the root login policy. Unspecified settings always default back to the root login policy. This means that a change to a root login policy option also affects all those policies for which the option was not specified.

All new databases include a root login policy. You can modify the root login policy values, but you cannot delete the policy. An overview of the default values for the root login policy is provided in the parameters section.

Privileges

You must have the `MANAGE ANY LOGIN POLICY` system privilege.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example creates the `Test1` login policy. This example has an unlimited password life and allows the user a maximum of five attempts to enter a correct password before the account is locked.


```
CREATE LOGIN POLICY Test1
PASSWORD_LIFE_TIME=UNLIMITED
MAX_FAILED_LOGIN_ATTEMPTS=5;
```

The following example shows typical settings for a new login policy (ldap_user_policy) that uses LDAP user authentication. Both a primary and a secondary server configuration object (previously created) are specified to allow failover to the secondary LDAP server, and the ability to failover to standard authentication is allowed when system resources, network resources, or, both primary and secondary LDAP servers are unresponsive. This example provides a combination of authentication options that permits responsiveness with cached values when an LDAP server cannot keep up with incoming requests. This example assumes that the login_mode database option includes 'Standard'. You cannot paste and run this example since the primary and secondary servers mentioned in the example are fictitious.

```
CREATE LOGIN POLICY ldap_user_policy
LDAP_PRIMARY_SERVER=ldapsrv1
LDAP_SECONDARY_SERVER=ldapsrv2
LDAP_FAILOVER_TO_STD=ON;
```

Related Information

[Login policies](#)

A **login policy** consists of a set of rules that are applied when you create a database connection for a user.

[Creating a new login policy](#)

Create custom login policies based on the root login policy.

[Assigning a login policy to an existing user](#)

Change the login policy that is assigned to a user by assigning the user a new login policy.

[ALTER LOGIN POLICY statement](#)

Alters an existing login policy.

[ALTER USER statement](#)

Alters user settings.

[COMMENT statement](#)

Stores a comment for a database object in the system tables.

[CREATE USER statement](#)

Creates a database user or group.

[DROP LOGIN POLICY statement](#)

Drops a login policy.

[DROP USER statement](#)

Drops a user.

[Login policy options and default values](#)

The following table lists the options that are governed by a login policy and includes the default values for the root login policy:

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CREATE MATERIALIZED VIEW statement

Creates a materialized view.

Syntax

CREATE MATERIALIZED VIEW

```
[ owner.]materialized-view-name [ ( alt-column-names, ... ) ]  
[ IN dbspace-name ]  
AS select-statement  
[ CHECK { IMMEDIATE | MANUAL } REFRESH ]
```

alt-column-names :
(column-name [,...])

Parameters

alt-column-names clause Use this clause to specify alternate names for the columns in the materialized view. If you specify alternate columns names, the number of columns listed in *alt-column-names* must match the number of columns in *select-statement*. If you do not specify alternate column names, the names are set to those in *select-statement*.

IN clause Use this clause to specify the dbspace in which to create the materialized view. If this clause is not specified, then the materialized view is created in the dbspace specified by the default_dbspace option. Otherwise, the system dbspace is used.

AS clause Use this clause to specify, in the form of a SELECT statement, the data to use to populate the materialized view. A materialized view definition can only reference base tables; it cannot reference views, other materialized views, or temporary tables. *select-statement* must contain column names or have an alias name specified. If you specify *alt-column-names*, those names are used instead of the aliases specified in *select-statement*.

Column names in the SELECT statement must be specified explicitly; you cannot use the [SELECT *](#) construct. For example, you cannot specify [CREATE MATERIALIZED VIEW matview AS SELECT * FROM table-name](#). Also, you should fully qualify objects names in the *select-statement*.

CHECK clause Use this clause to validate the statement without actually creating the view. When you specify the CHECK clause:

- The database server performs the normal language checks that would be carried out if CREATE MATERIALIZED VIEW was executed without the clause, and any errors generated are returned as usual.
- The database server does not perform the actual creation of the view, so certain errors that would occur at creation time are not generated. For example, an error indicating that the specified view name already exists is not generated. This allows you to use the CHECK clause to test intended changes to the definition of the view, without a conflict with the naming of the view.
- If CHECK IMMEDIATE REFRESH is used then the database server verifies that the syntax is valid for an immediate view and returns any errors.
- No changes are made to the database, and nothing is recorded in the transaction log.
- There is an implicit commit at the beginning of statement execution and a rollback at the end to release all locks obtained during execution.

Remarks

When you create a materialized view, it is a manual view and uninitialized. That is, it has a manual refresh type, and it has not been refreshed (populated with data). To initialize the view, execute a REFRESH MATERIALIZED VIEW statement, or use the sa_refresh_materialized_views system procedure.

You can encrypt a materialized view, change its PCTFREE setting, change its refresh type, and enable or disable its use by the optimizer. However, you must create the materialized view first, and then use the ALTER MATERIALIZED VIEW to change these settings. The default values for materialized views at creation time are:

- NOT ENCRYPTED
- ENABLE USE IN OPTIMIZATION
- PCTFREE is set according to the database page size: 200 bytes for a 4 KB page size, and 100 bytes for a 2 KB page size
- MANUAL REFRESH

Several database and server options must be in effect to create a materialized view.

The sa_recompile_views system procedure does not affect materialized views.

Privileges

You must have the CREATE MATERIALIZED VIEW system privilege to create materialized views owned by you. You must also be the owner of, or have SELECT privileges on, the underlying object referred to by the materialized view.

You must have the CREATE ANY MATERIALIZED VIEW or CREATE ANY OBJECT system privilege to create materialized views owned by others.

Side effects

Automatic commit. While executing, the CREATE MATERIALIZED VIEW statement places exclusive locks, without blocking, on all tables referenced by the materialized view. If one of the referenced tables cannot be locked, the statement fails and an error is returned.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example creates a materialized view containing confidential information about employees in the SQL Anywhere sample database. You must subsequently execute a REFRESH MATERIALIZED VIEW statement to initialize the view for use, as shown in the example.

```
CREATE MATERIALIZED VIEW EmployeeConfid2 AS
SELECT EmployeeID, Employees.DepartmentID,
       SocialSecurityNumber, Salary, ManagerID,
       Departments.DepartmentName, Departments.DepartmentHeadID
FROM GROUP0.Employees, GROUP0.Departments
WHERE Employees.DepartmentID=Departments.DepartmentID;
REFRESH MATERIALIZED VIEW EmployeeConfid2;
```

Related Information

[Materialized views](#)

A **materialized view** is a view whose result set has been precomputed from the base tables that it refers to and stored on disk, similar to a base table.

[Materialized views restrictions](#)

There are many restrictions when creating, initializing, refreshing, and using materialized views.

[Advanced: Status and properties for materialized views](#)

Materialized view availability and state can be determined from their status and properties.

[Additional dbspaces considerations](#)

Additional database files allow you to cluster related information in separate files.

[ALTER MATERIALIZED VIEW statement](#)

Alters a materialized view.

[DROP MATERIALIZED VIEW statement](#)

Removes a materialized view from the database.

[REFRESH MATERIALIZED VIEW statement](#)

Initializes or refreshes the data in a materialized view by executing its query definition.

[CREATE VIEW statement](#)

Creates a view on the database.

[sa_refresh_materialized_views system procedure](#)

Initializes all materialized views that are in an uninitialized state.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CREATE MESSAGE statement [T-SQL]

Creates a message number/message string pair. The message number can be used in PRINT and RAISERROR statements.

Syntax

CREATE MESSAGE *message-number* **AS** *message-text*

message-number : integer

message-text : string

Parameters

message-number The message number of the message to add. The message number for a user-defined message must be 20000 or greater.

message-text The text of the message to add. The maximum length is 255 bytes. PRINT and RAISERROR recognize placeholders in the message text. A single message can contain up to 20 unique placeholders in any order. These placeholders are replaced with the formatted contents of any arguments that follow the message when the text of the message is sent to the client.

The placeholders are numbered to allow reordering of the arguments when translating a message to a language with a different grammatical structure. A placeholder for an argument appears as "%nn!": a percent sign (%), followed by an integer from 1 to 20, followed by an exclamation mark (!), where the integer represents the position of the argument in the argument list. "%1!" is the first argument, "%2!" is the second argument, and so on.

There is no parameter corresponding to the *language* argument for sp_addmessage.

Remarks

Adds a user-defined message to the ISYSUSERMESSAGE system table for use by PRINT and RAISERROR statements.

Privileges

You must have the CREATE MESSAGE or CREATE ANY OBJECT system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.
- **Transact-SQL** CREATE MESSAGE supplies the functionality provided by the sp_addmessage system procedure in Adaptive Server Enterprise.

Example

The following example creates a new message:

```
CREATE MESSAGE 20000 AS 'End of line reached';
```

Related Information

[PRINT statement \[T-SQL\]](#)

Returns a message to the client, or display a message in the database server messages window.

[RAISERROR statement](#)

Signals an error and sends a message to the client.

[DROP MESSAGE statement](#)

Removes a message from the database.

[SYSUSERMESSAGE system view](#)

Each row in the SYSUSERMESSAGE system view holds a user-defined message for an error condition. The underlying system table for this view is ISYSUSERMESSAGE.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

CREATE MIRROR SERVER statement

Creates or replaces a mirror server that is being used for database mirroring or read-only scale-out.

Syntax

- **Create a mirror server**

```
CREATE [ OR REPLACE ] MIRROR SERVER mirror-server-name
AS { PRIMARY | MIRROR | ARBITER | PARTNER }
[ server-option = string [ ... ] ]
```

- **Create a mirror server as a copy**

```
CREATE [ OR REPLACE ] MIRROR SERVER mirror-server-name
AS COPY
{ FROM SERVER parent-name [ OR SERVER server-name ] | USING AUTO PARENT
}
[ server-option = string [ ... ] ]
```

server-option :

connection_string

logfile

preferred

state_file

parent-name :

server-name | **PRIMARY**

Parameters

OR REPLACE clause CREATE MIRROR SERVER creates the mirror server. An error is returned if a mirror server with the specified name already exists in the database.

Specifying OR REPLACE creates a mirror server if the server does not already exist in the database, and replaces it if it does exist.

AS clause You can specify one of the following server types:

- **PRIMARY** The mirror server with type PRIMARY defines a virtual or logical server, rather than an actual database server. The name of this server is the alternate server name for the database. The alternate server name can be used by applications to connect to the server currently acting as the primary server. The connection string for the server marked as PRIMARY
 - Defines the connection string used by copy nodes to connect to the root node or PRIMARY parent.
 - Defines the connection string used by the connection parameter NodeType PRIMARY value.

There can be only one PRIMARY server for a database.

- **MIRROR** The mirror server with type MIRROR defines a virtual or logical server, rather than an actual database server. The name of this server is the alternate mirror server name for the database. The alternate mirror server name can be used by applications to connect

to the server currently acting as the read-only mirror. The server marked as MIRROR also defines the connection string used by the NodeType connection parameter MIRROR value. There can be only one MIRROR server for a database.

- **ARBITER** In a database mirroring system, the arbiter server assists in determining which of the PARTNER servers takes ownership of the database. The arbiter server must be defined with a connection string that can be used by the partner servers to connect to the arbiter. There can be only one ARBITER server for a database.
- **PARTNER** The name of the mirror server must correspond to the name of the database server, as specified by the -n server option, and must match the value of the SERVER connection string parameter specified in the connection_string mirror server option.

In a database mirroring system, the partners use the connection string value to connect to each other. In a read-only scale-out system, the connection string is used by a copy-node that has this server as its parent.

- **Mirroring or mirroring with read-only scale-out** You must define two PARTNER servers for database mirroring, and both must have a connection string and a state file.

In a database mirroring system, servers defined as PARTNER are eligible to become the primary server and take ownership of the database.

- **Read-only scale-out without mirroring** You must define one PARTNER server for read-only scale-out, and it must have a connection string and no state file. This server is the root server, and runs the only copy of the database that allows both read and write operations
- **COPY** In a read-only scale-out system, this value specifies that the database server is a copy node. All connections to the database on this server are read-only. The name of the mirror server must correspond to the name of the database server, as specified by the -n server option, and must match the value of the SERVER connection string parameter specified in the connection_string mirror server option.

When AS COPY is specified, then you must also specify either the FROM SERVER or USING AUTO PARENT clause.

The connection string is used by the NodeType connection parameter COPY value and it is also used by other copy nodes that have this server as their parent.

When adding copy nodes to a read-only scale-out system, you can either execute a CREATE MIRROR SERVER statement for the copy node, or have the root server define the mirror server automatically.

FROM SERVER clause This clause can only be used when AS COPY is specified. This clause constructs a tree of servers for a scale-out system and indicates which servers the copy nodes obtain transaction log pages from.

The parent can be specified using the name of the mirror server or PRIMARY. An alternate parent for the copy node can be specified using the OR SERVER clause.

In a database mirroring system that has only two levels (partner and copy nodes), the copy nodes obtain transaction log pages from the current primary or mirror server.

A copy node determines which server to connect to by using its mirror server definition that is stored in the database. From its definition, it can locate the definition of its parent, and from its parent's definition, it can obtain the connection string to connect to the parent.

You do not have to explicitly define copy nodes for the scale-out system: you can choose to have the root node define the copy nodes when they connect.

OR SERVER clause Use the OR SERVER clause to specify an alternate parent for the copy node.

USING AUTO PARENT clause This clause can only be used when AS COPY is specified. This clause causes the primary server to assign a parent for this server. When you use this clause to replace an existing copy node server, the definitions for the parent and alternate parent for the copy node do not change.

server-option clause You can specify a variable name for *server-option*.

The following options are supported:

- **connection_string server option** Specifies the connection string to be used to connect to the server. The connection string for a mirror server should not include a user ID or password because they are not used when one mirror server connects to another mirror server.
- **logfile server option** Specifies the location of the file that contains one line per request that is sent between mirror servers if database mirroring is used. This file is used only for debugging.
- **preferred server option** Specifies whether the server is the preferred server in the mirroring system. You can specify either YES or NO. The preferred server assumes the role of primary server whenever possible. You specify this option when defining PARTNER servers.
- **state_file server option** Specifies the location of the file used for maintaining state information about the mirroring system. This option is required for database mirroring. In a mirroring system, a state file must be specified for servers with type PARTNER. For arbiter servers, the location is specified as part of the command to start the server.

Remarks

Read-only scale-out and database mirroring each require a separate license.

This statement creates or replaces a mirror server definition; it does not change a mirror server definition. To change a mirror server definition, use the ALTER MIRROR SERVER statement.

In a database mirroring system, the mirror server type can be PRIMARY, MIRROR, ARBITER, or PARTNER.

In a read-only scale-out system, the mirror server type can be PRIMARY, PARTNER, or COPY.

Mirror server names for servers of type PARTNER or COPY must match the names of the database servers that are part of the mirroring system (the name used with the -n server option). This requirement allows each database server to find its own definition and that of its parent. Also, all copy node servers must have unique server names. *mirror-server-name*, *parent-name*, and *server-name* above must be 7-bit ASCII characters.

To use a copy node as the arbiter for the database it is copying in a database mirroring system, create the arbiter server with a name that does not match the server name of any of the database servers in the high availability system. In this configuration, the name of the arbiter is used as a placeholder in the mirror server definition to hold the connection string for the arbiter.

Note For parameters that accept variable names, an error is returned if one of the following conditions is true:

- The variable does not exist
- The contents of the variable are NULL
- The variable exceeds the length allowed by the parameter
- The data type of the variable does not match that required by the parameter

Privileges

You must have the `MANAGE ANY MIRROR SERVER` system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statement creates a mirror server that can be used as the primary server in a database mirroring system:

```
CREATE MIRROR SERVER "scaleout_primary"  
AS PRIMARY  
connection_string = 'server=scaleout_primary;host=winxp-2:6871,winxp-3:6872';
```

The following statement creates a mirror server that can be used as the mirror server in a database mirroring system:

```
CREATE MIRROR SERVER "scaleout_mirror"  
AS MIRROR  
connection_string = 'server=scaleout_mirror;host=winxp-2:6871,winxp-3:6872';
```

The following statement creates a mirror server that can be used as the arbiter in a database mirroring system:

```
CREATE MIRROR SERVER "scaleout_arbiter"  
AS ARBITER  
connection_string = 'server=scaleout_arbiter;host=winxp-4:6870';
```

The following statement creates two mirror servers that can be used as partners server in a database mirroring system:

```
CREATE MIRROR SERVER "scaleout_server1"  
AS PARTNER  
connection_string = 'server=scaleout_server1;HOST=winxp-2:6871'  
state_file = 'c:\\server1\\server1.state';
```

```
CREATE MIRROR SERVER "scaleout_server2"  
AS PARTNER  
connection_string = 'server=scaleout_server2;HOST=winxp-3:6872'  
state_file = 'c:\\server2\\server2.state';
```

The following statement creates a copy node that can act as the arbiter in a database mirroring system:

```
CREATE MIRROR SERVER "scaleout_child"  
AS COPY FROM SERVER "scaleout_primary"  
connection_string = 'server=scaleout_child;host=winxp-5:6878';
```

The following statement defines a copy node as the arbiter for a different database mirroring system:

```
CREATE MIRROR SERVER "The Arbiter"  
AS ARBITER  
connection_string = 'server=scaleout_child;host=winxp-5:6878';
```

The following statement preserves the current parent if *server-name* already exists. However, it does not auto-generate a new parent.

```
CREATE OR REPLACE MIRROR SERVER "server-name" AS COPY USING AUTO PARENT;
```

Example

The following example creates a mirror server using a variable for the *connection_string* parameter.

The following statement creates a variable for a connection string:

```
CREATE VARIABLE @connstr_value LONG VARCHAR ;  
SET @connstr_value = ' server=new_scaleout_primary;host=winxp-2:6871,winxp-3:6872'  
' ;
```

The following statement creates a mirror server using the variable @connstr_value for the *connection_string* parameter:

```
CREATE MIRROR SERVER new_scaleout_primary AS PRIMARY  
connection_string = @connstr_value ;
```

Related Information

[Connection parameters](#)

Connection parameters are included in connection strings.

[Database mirroring](#)

Database mirroring is a configuration of three database servers, running on separate computers, that co-operate to maintain copies of the database and transaction log files.

[Read-only scale-out](#)

Read-only scale-out allows you to offload reporting or other operations that require read-only access to the database.

[How child copy nodes are added](#)

The child_creation option of the SET MIRROR OPTION statement controls how child nodes are added to a read-only scale-out system.

[Troubleshooting: State information files of the partners and arbiter](#)

The partners and the arbiter in the high availability system each maintain a state information file that records that server view of the state of the mirroring system.

[Preferred database server in a database mirroring system](#)

In a database mirroring system, you can specify one partner server as the preferred server.

[Automatically assign the parent of a copy node](#)

The SET MIRROR OPTION statement supports two options that can be used to assign new copy nodes to a parent in the tree so that the work of distributing transaction log pages is balanced among the nodes.

[Separately licensed components](#)

These components are licensed separately and may need to be ordered separately if not included in your edition of SQL Anywhere.

[SYSMIRRORSERVER system view](#)

The underlying system table for this view is ISYSMIRRORSERVER.

[SET MIRROR OPTION statement](#)

Changes the values of options that control the settings for database mirroring and read-only scale-out.

[ALTER MIRROR SERVER statement](#)

Modifies the attributes of a mirror server.

[COMMENT statement](#)

Stores a comment for a database object in the system tables.

[DROP MIRROR SERVER statement](#)

Drops a mirror server.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

CREATE MUTEX statement

Creates or replaces a mutex (lock) that can be used to lock a resource such as a file or a procedure.

Syntax

```
CREATE [ OR REPLACE | TEMPORARY ] MUTEX [ IF NOT EXISTS ] [ owner.]mutex-name  
[ SCOPE { CONNECTION | TRANSACTION } ]
```

Parameters

- **owner** The owner of the mutex. *owner* can also be specified using an indirect identifier (for example, ``[@variable-name]``).
- **mutex-name** The name of the mutex. Specify a valid identifier in the CHAR database collation. *semaphore-name* can also be specified using an indirect identifier (for example, ``[@variable-name]``).
- **OR REPLACE clause** Use this clause to overwrite (update) the definition of a permanent mutex of the same name, if one exists.

If the OR REPLACE clause is specified, and a mutex with this name is in use at the time, then the statement returns an error.

Do not use this clause with the TEMPORARY or IF NOT EXISTS clauses.
- **TEMPORARY clause** Use this clause to create a temporary mutex.

Do not use this clause with the OR REPLACE clause.
- **IF NOT EXISTS clause** Use this clause to create a mutex only if it doesn't already exist. If a mutex exists with the same name, then nothing happens and no error is returned.

Do not use this clause with the OR REPLACE clause.
- **SCOPE clause** Use this clause to specify whether the mutex applies to a transaction (TRANSACTION), or the connection (CONNECTION). If the SCOPE clause is not specified, then the default behavior is CONNECTION.

Remarks

Permanent and temporary mutexes and semaphores share the same namespace, therefore you cannot create two of these objects with the same name. Use of the OR REPLACE and IF NOT EXISTS clause can inadvertently cause an error related to naming. For example, if you have a permanent mutex, and you try to create a temporary semaphore with the same name, an error is returned even if you specify IF NOT EXISTS. Similarly, if you have a temporary semaphore, and you try to replace it with a permanent semaphore with the same name by specifying OR REPLACE, an error is returned because this is equivalent to attempting to create a second object with the same name.

Permanent mutex definitions persist across database restarts. However, their state information (locked or released), does not.

A temporary mutex persists until the connection that created it is terminated, or until the mutex is dropped using a DROP MUTEX statement. If another connection is waiting for a temporary mutex and the connection that created the temporary mutex is terminated, then an error is returned to the waiting connection indicating that the mutex has been deleted.

A best practice is to avoid the use of the IF NOT EXISTS clause with the OR REPLACE clause.

CONNECTION scope mutexes are not automatically released other than when the connection is terminated.

Privileges

You must have the CREATE ANY MUTEX SEMAPHORE or CREATE ANY OBJECT system privilege.

Side effects

Automatic commit, but only for permanent mutexes.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statement creates a connection scope mutex called protect_my_cr_section to protect a critical section of a stored procedure.

```
CREATE MUTEX protect_my_cr_section SCOPE CONNECTION;
```

Related Information

[Mutexes and semaphores](#)

Use mutexes and semaphores in your application logic to achieve locking behavior and control and communicate the availability of resources.

[DROP MUTEX statement](#)

Drops the specified mutex.

[LOCK MUTEX statement](#)

Locks a resource such as a file or system procedure using a predefined mutex.

[RELEASE MUTEX statement](#)

Releases the specified connection-scope mutex, if it is locked by the current connection.

[SYSMUTEXSEMAPHORE system view](#)

Each row in the SYSMUTEXSEMAPHORE system view provides information about a user-defined mutex or semaphore in the database. The underlying system table for this view is ISYSMUTEXSEMAPHORE.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

CREATE ODATA PRODUCER statement

Creates or replaces an OData Producer.

Syntax

CREATE [OR REPLACE] ODATA PRODUCER *name* [*producer-clause...*]

producer-clause:

[**ADMIN USER** { **NULL** | *user* }]

[**AUTHENTICATION** [**DATABASE** | **USER** *user*]]

[[**NOT**] **ENABLED**]

[**MODEL** [**FROM**] { **FILE** *string-or-variable* [**ENCODING** *string*] | **VALUE** { **NULL** | *string-or-variable* [**ENCODING** *string*] } }]

[[**SERVICE**] **ROOT** *string*]

[**USING** *string*]

Parameters

- **ADMIN USER clause** Specifies an administrative user that the OData Producer uses to create tables, a procedure, and an event to manage repeatable requests. The user must have the CREATE TABLE, CREATE PROCEDURE, VERIFY ODATA, and MANAGE ANY EVENT system privileges. This user cannot be the same user as specified in the AUTHENTICATION clause.
- **AUTHENTICATION clause** Specifies how the OData server connects to the database. The default setting, DATABASE, allows users to connect with personalized credentials. These credentials are requested using Basic HTTP authentication. Specify USER for all users to connect using the credentials of the specified user. The specified user must have the VERIFY ODATA system privilege.
- **ENABLED clause** Specifies whether this OData Producer is enabled or disabled. By default, OData Producers are enabled.
- **MODEL clause** Specifies the OData Producer service model (given in OSDL), that indicates which tables and views are exposed in the OData metadata. If you specify the FILE clause, then the value must be a variable or string of the file name that contains the OSDL data. If the file name is not fully qualified, then the path is relative to the database server's current working directory. If the file name is specified, then the file is read and the contents of the file are stored in the database and logged in the transaction log. By default, tables and views are exposed based on user privileges. Tables and views without primary keys are not exposed.

Use the ENCODING clause to specify the character set of the model data or file. The default is UTF-8.

- **SERVICE ROOT clause** Specifies the root of the OData service on the OData server. All resources for the OData Producer are accessed by using URIs of the following format:
scheme:host:port/path-prefix/resource-path[query-options].
 The SERVICE ROOT clause is /path-prefix. The value cannot be NULL and must start with /.
- **USING clause** Specifies additional OData Producer options. Options are specified in a semicolon-separated list of name=value pairs.

Option	Description
ConnectionPoolMaximum = <i>num-max-connections</i>	<p>Indicates the maximum number of simultaneous connections that this OData Producer keeps open for use in the connection pool.</p> <p>Fewer connections might be used by the connection pool depending on the OData server load.</p> <p>By default, the connection pool size is limited to half of the maximum number of simultaneous connections that the database server supports.</p>
CSRFTokenTimeout = <i>num-seconds-valid</i>	<p>Enables CSRF token checking and specifies the number of seconds that a token is valid for.</p> <p>By default, this value is 0, which disables CSRF token checking. Otherwise, the number of seconds must be a valid integer value from 1 to 1800.</p>
PageSize = <i>num-max-entities</i>	<p>Specifies the maximum number of entities to include in a retrieve entity set response before issuing a next link.</p> <p>The default setting is 100.</p>
ReadOnly = { true false }	<p>Indicates whether modification requests should be ignored.</p> <p>The default setting is false.</p>
RepeatRequestForDays = { <i>days-number</i> }	<p>Specifies how long repeatable requests are valid.</p> <p>This value must be an integer ranging from 1 to 31.</p> <p>The default setting is 2.</p> <p>This option is only effective when the ADMIN USER clause is not NULL.</p>
SecureOnly = { true false }	<p>Indicates whether the Producer should only listen for requests on the HTTPS port.</p> <p>The default setting is false.</p>
ServiceOperationColumnNames = { generate database }	<p>Specifies whether the column names in the metadata should be generated or taken from the result set columns in the database when naming the properties of the ComplexType used in the ReturnType.</p> <p>The default setting is generate.</p>

Remarks

Creates an OData Producer.

When a database is started on a database server that was started with the -xs ODATA database server option, then all enabled OData Producers are started on the database server.

Privileges

You must have the MANAGE ODATA system privilege.

If you specify the AUTHENTICATION USER clause or the ADMIN USER clause, then the specified users must have the VERIFY ODATA system privilege.

Side effects

If the OData Producer is running, then it may be stopped and restarted.

Example

The following example shows how to create an OData Producer:

```
CREATE ODATA PRODUCER OrderEntryProducer
  ADMIN USER SuperODataUser
  AUTHENTICATION USER ODataUser
  ENABLED
  MODEL FILE '../ordermodel.osdl'
  SERVICE ROOT '/orders/'
  USING 'ConnectionPoolMaximum=10;CSRFTokenTimeout=600;PageSize=100;
  ReadOnly=false;RepeatRequestForDays=3;SecureOnly=true;
  ServiceOperationColumnNames=generate';
```

Related Information

[OData support](#)

OData (Open Data Protocol) enables data services over RESTful HTTP. It allows you to perform operations through URIs (Uniform Resource Identifiers) to access and modify information.

[Network protocol options](#)

Network protocol options enable you to work around traits of different network protocol implementations.

[ALTER ODATA PRODUCER statement](#)

Alters an OData Producer.

[DROP ODATA PRODUCER statement](#)

Drops an OData Producer.

[COMMENT statement](#)

Stores a comment for a database object in the system tables.

[-xs database server option](#)

Specifies server-side web services communications protocols.

[SYSODATAPRODUCER system view](#)

Each row of the SYSODATAPRODUCER system view describes an OData Producer. The underlying system table for this view is ISYSODATAPRODUCER.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

CREATE PROCEDURE statement [External call]

Creates an interface to a native or external procedure.

Syntax

```
CREATE [ OR REPLACE ] PROCEDURE [ owner.]procedure-name
  ( [ parameter[, ... ] ] )
[ RESULT ( result-column [, ... ] )
  | NO RESULT SET
  | DYNAMIC RESULT SETS integer-expression ]
[ SQL SECURITY { INVOKER | DEFINER } ]
{ EXTERNAL NAME 'native-call'
  | EXTERNAL NAME 'c-call' LANGUAGE { C_ESQL32 | C_ESQL64 | C_ODBC32 |
C_ODBC64 }
  | EXTERNAL NAME 'clr-call' LANGUAGE CLR
  | EXTERNAL NAME 'perl-call' LANGUAGE PERL
  | EXTERNAL NAME 'php-call' LANGUAGE PHP
  | EXTERNAL NAME 'java-call' LANGUAGE JAVA
  | EXTERNAL NAME 'js-call' LANGUAGE JS }
```

parameter :

```
[ parameter-mode ] parameter-name data-type [ DEFAULT expression ]
| SQLCODE
| SQLSTATE
```

parameter-mode :

```
IN
| OUT
| INOUT
```

result-column :

column-name *data-type*

native-call :

```
[ system-configuration: ]function-name@library-file-prefix[.{ so | dll } ]
```

system-configuration :

```
{ generic-operating-system | specific-operating-system } [ ( processor-architecture ) ]
```

generic-operating-system :

```
{ Unix | Windows }
```

specific-operating-system :

```
{ AIX | HPUX | Linux | OSX | Solaris | WindowsNT }
```

processor-architecture :

```
{ 32 | 64 | ARM | IA64 | PPC | SPARC | X86 | X86_64 }
```

c-call :

[*operating-system:*] *function-name@library*; ...

operating-system :

Unix

clr-call :

dll-name::function-name(param-type-1[, ...])

perl-call :

<file=perl-file> \$sa_perl_return = perl-subroutine(\$sa_perl_arg0[, ...])

php-call :

<file=php-file> print php-func(\$argv[1][, ...])

java-call :

[*package-name.*] *class-name.method-name java-method-signature*

java-method-signature :

([*java-field-descriptor*, ...]) *java-return-descriptor*

java-field-descriptor and *java-return-descriptor :*

```

{ Z
  | B
  | S
  | I
  | J
  | F
  | D
  | C
  | V
  | [ descriptor
  | L class-name;
}

```

js-call :

<js-return-descriptor> <file=js-object> js-func(js-field-descriptor[...])

js-field-descriptor and *js-return-descriptor :*

```

{ S
  | B
  | I
  | U
  | D
  | [ descriptor
}

```

Parameters

You can create permanent stored procedures that call external or native procedures written in a variety of programming languages. You can use PROC as a synonym for PROCEDURE.

OR REPLACE clause Specifying CREATE OR REPLACE PROCEDURE creates a new procedure, or replaces an existing procedure with the same name. This clause changes the definition of the procedure, but preserves existing privileges. An error is returned if you attempt to replace a procedure that is already in use.

parameter Parameter names must conform to the rules for other database identifiers such as column names. They must be a valid SQL data type.

Parameters can be prefixed with one of the keywords IN, OUT, or INOUT. If you do not specify one of these values, then parameters are INOUT by default. The keywords have the following meanings:

- **IN** The parameter is an expression that provides a value to the procedure.
- **OUT** The parameter is a variable that could be given a value by the procedure.
- **INOUT** The parameter is a variable that provides a value to the procedure, and could be given a new value by the procedure.

You can set the data type explicitly, or specify the %TYPE or %ROWTYPE attribute to set the data type to the data type of another object in the database. Use %TYPE to set it to the data type of a column in a table or view. Use %ROWTYPE to set the data type to a composite data type derived from a row in a table or view.

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, then missing parameters are assigned the default values. If an argument is not provided in the CALL statement, and no default is set, then an error is given.

SQLSTATE and SQLCODE are special OUT parameters that output the SQLSTATE or SQLCODE value when the procedure ends. The SQLSTATE and SQLCODE special values can be checked immediately after a procedure call to test the return status of the procedure.

The SQLSTATE and SQLCODE special 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.

Specifying OR REPLACE (CREATE OR REPLACE PROCEDURE) creates a new procedure, or replaces an existing procedure with the same name. This clause changes the definition of the procedure, but preserves existing privileges. An error is returned if you attempt to replace a procedure that is in use.

You cannot create TEMPORARY external call procedures.

RESULT clause The RESULT clause declares the number and type of columns in the result set. The parenthesized list following the RESULT keyword defines the result column names and types. This information is returned by the Embedded SQL DESCRIBE or by ODBC SQLDescribeCol when a CALL statement is being described.

If a RESULT clause is specified, then it must be the first clause of the statement.

Embedded SQL (LANGUAGE C_ESQL32, LANGUAGE C_ESQL64) or ODBC (LANGUAGE C_ODBC32, LANGUAGE C_ODBC64) external procedures can return 0 or 1 result sets.

Perl, PHP (LANGUAGE PERL, LANGUAGE PHP), or JavaScript external procedures cannot return result sets. Procedures that call native functions loaded by the database server cannot return result sets.

CLR or Java (LANGUAGE CLR, LANGUAGE JAVA) external procedures can return 0, 1, or more result sets.

Some procedures produce more than one result set, 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 GivenName
    FROM GROUP0.Employees
  ELSE
    SELECT Surname, GivenName
    FROM Employees
  END IF
END;
```

Procedures with variable result sets 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, to get the proper shape of result set. The CURSOR *cursor-name* clause on the DESCRIBE statement is required.
- **ODBC, OLE DB, ADO.NET** Variable result-set procedures can be used by applications using these interfaces. The proper description of the result sets is carried out by the driver or provider.
- **Open Client applications** Variable result-set procedures can be used by Open Client applications.

If your procedure returns only one result set, then 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.

To handle multiple result sets, ODBC must describe the currently executing cursor, not the procedure's defined result set. Therefore, ODBC does not always describe column names as defined in the RESULT clause of the procedure definition. To avoid this problem, use column aliases in the SELECT statement that generates the result set.

NO RESULT SET clause If a NO RESULT SET clause is specified, then it must be the first clause of the statement.

Declares that no result set is returned by this procedure. This declaration can lead to a performance improvement.

DYNAMIC RESULT SETS clause If a DYNAMIC RESULT SETS clause is specified, it must be the first clause of the statement.

Use this clause with LANGUAGE CLR and LANGUAGE JAVA calls. The DYNAMIC RESULT SETS clause is used to specify the number of dynamic result sets that will be returned by the procedure. When a RESULT clause is specified and the DYNAMIC RESULT SETS clause is not specified, it is assumed that the number of dynamic result sets is 1. When neither the RESULT clause nor the DYNAMIC RESULT SETS clause is specified, no result set is expected and an error will result if a result set is generated.

The C_ESQL32, C_ESQL64, C_ODBC32, and C_ODBC64 external environments can also return result sets (like CLR and JAVA), but they are restricted to only one dynamic result set.

Procedures that call into Perl, PHP (LANGUAGE PERL, LANGUAGE PHP), or JavaScript external

functions cannot return result sets. Procedures that call native functions loaded by the database server cannot return result sets.

SQL SECURITY clause The SQL SECURITY clause defines whether the procedure is executed as the INVOKER (the user who is calling the procedure), or as the DEFINER (the user who owns the procedure). The default is DEFINER. For external calls, this clause establishes the ownership context for unqualified object references in the external environment.

When SQL SECURITY INVOKER is specified, more memory is used because annotation must be done for each user that calls the procedure. Also, when SQL SECURITY INVOKER is specified, name resolution is done as the invoker as well. Therefore, qualify all object names (tables, procedures, and so on) with their appropriate owner. For example, suppose user1 creates the following procedure:

```
CREATE PROCEDURE user1.myProcedure()  
  RESULT( columnA INT )  
  SQL SECURITY INVOKER  
  BEGIN  
    SELECT columnA FROM table1;  
  END;
```

If user2 attempts to run this procedure and a table user2.table1 *does not* exist, a table lookup error results. Additionally, if a user2.table1 *does* exist, that table is used instead of the intended user1.table1. To prevent this situation, qualify the table reference in the statement (user1.table1, instead of just table1).

EXTERNAL NAME *native-call* clause Because *native-call* can specify multiple sets of operating systems, processors, libraries, and functions, more-precisely specified configurations take precedence over less-precisely defined configurations. For example, `Solaris(X86_64):myfunc64@mylib.so` takes precedence over `Solaris:myfunc64@mylib.so`.

For syntaxes that support *system-configuration*, if you do not specify *system-configuration*, then it is assumed that the procedure runs on all system configurations. Unix represents the following Unix-based operating systems: AIX, HP-UX, Linux, Mac OS X, and Solaris. The generic term Windows represents all versions of the Windows operating system.

If you specify Unix for one of the calls, then it is assumed that the other call is for Windows.

The *specific-operating-system* and *processor-architecture* values are those operating systems and processors supported by SQL Anywhere Server.

The library name (*library-file-prefix*) is followed by the file extension, which is typically `.dll` on Windows and `.so` on Unix. In the absence of the extension, the software appends the platform-specific default file extension for libraries. For example:

```
CREATE PROCEDURE mystring( IN instr LONG VARCHAR )  
  EXTERNAL NAME 'mystring@mylib.dll;Unix:mystring@mylib.so';
```

A simpler way to write the EXTERNAL NAME clause, using platform-specific defaults, is as follows:

```
CREATE PROCEDURE mystring( IN instr LONG VARCHAR )  
  EXTERNAL NAME 'mystring@mylib';
```

When called, the library containing the function is loaded into the address space of the database server. The native function executes as part of the database server. In this case, if the function causes a fault, then the database server shuts down. Because of this behavior, loading and executing functions in an external environment using the LANGUAGE attribute is recommended. If a function causes a fault in an external environment, then the database server continues to run.

- **EXTERNAL NAME *c-call* clause** To call a compiled native C function in an external environment instead of within the database server, the stored procedure or function is defined with the EXTERNAL NAME clause followed by the LANGUAGE attribute.

When the LANGUAGE attribute is specified, then the library containing the function is loaded by an external process and the external function executes as part of that external process. In this case, if the function causes a fault, then the database server continues to run.

The following is a sample procedure definition.

```
CREATE PROCEDURE ODBCinsert(  
    IN ProductName CHAR(30),  
    IN ProductDescription CHAR(50)  
)  
NO RESULT SET  
EXTERNAL NAME 'ODBCexternalInsert@extodbc.dll'  
LANGUAGE C_ODBC32;
```

- **EXTERNAL NAME *clr-call* clause** To call a .NET function in an external environment, the procedure interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE CLR attribute.

A CLR stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in a .NET language such as C# or Visual Basic, and the execution of the procedure or function takes place outside the database server (that is, within a separate .NET executable).

```
CREATE PROCEDURE clr_interface(  
    IN p1 INT,  
    IN p2 UNSIGNED SMALLINT,  
    OUT p3 LONG VARCHAR)  
NO RESULT SET  
EXTERNAL NAME 'CLRlib.dll::CLRproc.Run( int, ushort, out string )'  
LANGUAGE CLR;
```

- **EXTERNAL NAME *perl-call* clause** To call a Perl function in an external environment, the procedure interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE PERL attribute.

A Perl stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in Perl and the execution of the procedure or function takes place outside the database server (that is, within a Perl executable instance).

The following is a sample procedure definition.

```
CREATE PROCEDURE PerlWriteToConsole( IN str LONG VARCHAR)
NO RESULT SET
EXTERNAL NAME '<file=PerlConsoleExample>
    WriteToServerConsole( $sa_perl_arg0 )'
LANGUAGE PERL;
```

- **EXTERNAL NAME *php-call* clause** To call a PHP function in an external environment, the procedure interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE PHP attribute.

A PHP stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in PHP and the execution of the procedure or function takes place outside the database server (that is, within a PHP executable instance).

The following is a sample procedure definition.

```
CREATE PROCEDURE PHPPopulateTable()
NO RESULT SET
EXTERNAL NAME '<file=ServerSidePHPEXample> ServerSidePHPSub()'
LANGUAGE PHP;
```

- **EXTERNAL NAME *java-call* clause** To call a Java method in an external environment, the procedure interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE JAVA attribute.

A Java-interfacing stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in Java and the execution of the procedure or function takes place outside the database server (that is, within a Java VM).

The following is a sample procedure definition.

```
CREATE PROCEDURE HelloDemo( IN name LONG VARCHAR )
NO RESULT SET
EXTERNAL NAME 'Hello.main([Ljava/lang/String;)V'
LANGUAGE JAVA;
```

The descriptors for arguments and return values from Java methods have the following meanings:

Field type	Java data type
B	byte
C	char
D	double
F	float
I	int
J	long
L <i>class-name</i> ;	An instance of the class <i>class-name</i> . The class name must be fully qualified, and

Field type	Java data type
	any dot in the name must be replaced by a /. For example, java/lang/String .
S	short
V	void
Z	Boolean
[Use one for each dimension of an array.

- **EXTERNAL NAME *js-call* clause** To call a JavaScript function in an external environment, the procedure interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE JS attribute.

A JavaScript stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in JavaScript and the execution of the procedure or function takes place outside the database server (that is, within a Node.js executable instance).

Specify the return type of the JavaScript function at the beginning of the EXTERNAL NAME string inside angle brackets. Since JavaScript does not allow pass-by-reference for simple variables inside functions, the left bracket character ([) can proceed the S, B, I, U, or D characters to indicate that a one element array is being passed to the JavaScript stored procedure. This syntax is provided to support INOUT and OUT parameters in stored procedures.

The following is a sample procedure definition.

```
CREATE PROCEDURE JSInOutDemo( INOUT num1 INT, OUT num2 INT )
  EXTERNAL NAME '<file=JSInOutParam> JSFunctionPlusOne([I[I])'
LANGUAGE JS;
```

The descriptors for arguments and return values from JavaScript methods have the following meanings:

Field type	JavaScript data type
S	String
B	Boolean
I	Integer
U	Unsigned integer
D	Double

Remarks

Clause order is important for the following clauses, which, when specified, must appear in the order listed here:

- Result-related clauses (RESULT, NO RESULT SET, DYNAMIC RESULT SETS)
- SQL SECURITY
- EXTERNAL NAME

The CREATE PROCEDURE statement creates a procedure in the database. You can create procedures for other users by specifying an owner. A procedure is invoked with a CALL statement.

If a stored procedure returns a result set, it cannot also set output parameters or return a return

value.

When referencing a temporary table from multiple procedures, a potential issue can arise if the temporary table definitions are inconsistent and statements referencing the table are cached.

Privileges

You must have the CREATE PROCEDURE system privilege to create external procedures you own.

You must have the CREATE ANY PROCEDURE or CREATE ANY OBJECT system privilege to create external procedures owned by others.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** CREATE PROCEDURE for an external language environment is a core feature of the ANSI/ISO SQL Standard, though some of its components supported in the software are optional SQL Language Features. A subset of these features include:

- The SQL SECURITY clause is optional Language Feature T324.
- The ability to pass a LONG VARCHAR, LONG NVARCHAR, or LONG BINARY value to an external procedure is SQL Language Feature T041.
- The ability to create or modify a schema object within an external procedure, using statements such as CREATE TABLE or DROP TRIGGER, is SQL Language Feature T653.
- The ability to use a dynamic-SQL statement within an external procedure, including statements such as CONNECT, EXECUTE IMMEDIATE, PREPARE, and DESCRIBE, is SQL Language Feature T654.
- JAVA external procedures embody SQL Language Feature J621.

Several clauses of the CREATE PROCEDURE statement are not in the standard. These include:

- Support for C_ESQL32, C_ESQL64, C_ODBC32, C_ODBC64, CLR, PERL, and PHP in the LANGUAGES clause are not in the standard. The ANSI/ISO SQL Standard supports "C" as an *environment-name* as optional Language Feature B122.
 - The format of *external-call* is implementation-defined.
 - The RESULT and NO RESULT SET clauses are not in the standard. The ANSI/ISO SQL Standard uses the RETURNS clause.
 - The optional DEFAULT clause for a specific routine parameter is not in the standard.
 - The optional OR REPLACE clause is not in the standard.
- **Transact-SQL** CREATE PROCEDURE for an external routine is supported by Adaptive Server Enterprise. Adaptive Server Enterprise supports C-language and Java language external routines.

Related Information

[The JavaScript external environment](#)

JavaScript stored procedures and functions can be called from the database in the same manner as user-defined SQL stored procedures and functions.

[References to temporary tables within procedures](#)

Sharing a temporary table between procedures can cause problems if the table definitions are

inconsistent.

[The ESQL and ODBC external environments](#)

External compiled native functions that use Embedded SQL or ODBC can be called from the database in the same manner as SQL stored procedures.

[External call interface](#)

You can call a function in an external library from a stored procedure or function.

[External environment support](#)

Six external runtime environments are supported. These include Embedded SQL and ODBC applications written in C/C++, and applications written in Java, JavaScript, Perl, PHP, or languages such as C# and Visual Basic that are based on the Microsoft .NET Framework Common Language Runtime (CLR).

[The Perl external environment](#)

Perl stored procedures and functions can be called from the database in the same manner as SQL stored procedures.

[The CLR external environment](#)

CLR stored procedures and functions can be called from within the database in the same manner as SQL stored procedures.

[The PHP external environment](#)

PHP stored procedures and functions can be called from the database in the same manner as SQL stored procedures.

[The Java external environment](#)

Java methods can be called from the database in the same manner as SQL stored procedures.

[%TYPE and %ROWTYPE attributes](#)

In addition to explicitly setting the data type for an object, you can also set the data type by specifying the %TYPE and %ROWTYPE attributes.

[ALTER PROCEDURE statement](#)

Modifies a procedure.

[CALL statement](#)

Invokes a procedure.

[CREATE FUNCTION statement](#)

Creates a user-defined SQL function in the database.

[CREATE FUNCTION statement \[External call\]](#)

Creates an interface to a native or external function.

[CREATE PROCEDURE statement](#)

Creates a user-defined SQL procedure in the database.

[CREATE PROCEDURE statement \[Web service\]](#)

Creates a user-defined web client procedure that makes HTTP or SOAP requests to an HTTP server.

[CREATE PROCEDURE statement \[T-SQL\]](#)

Creates a new procedure in the database in a manner compatible with Adaptive Server Enterprise.

[DROP PROCEDURE statement](#)

Removes a procedure from the database.

[GRANT statement](#)

Grant system and object-level privileges to users and roles.

[SQL data types](#)

There are many SQL data types supported by the software.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)» [Alphabetical list of SQL statements](#)

CREATE PROCEDURE statement [Web service]

Creates a user-defined web client procedure that makes HTTP or SOAP requests to an HTTP server.

Syntax

```

CREATE [ OR REPLACE ] PROCEDURE [ owner. ] procedure-name ( [ parameter, ... ] )
[ RESULT ( attribute-column-name datatype, value-column-name datatype ) ]
URL url-string
[ TYPE { http-type-spec-string | soap-type-spec-string } ]
[ HEADER header-string ]
[ CERTIFICATE certificate-string ]
[ CLIENTPORT clientport-string ]
[ PROXY proxy-string ]
[ SET protocol-option-string ]
[ SOAPHEADER soap-header-string ]
[ NAMESPACE namespace-string ]

```

http-type-spec-string :

```

HTTP[: { GET
| POST[:MIME-type ]
| PUT[:MIME-type ]
| DELETE
| HEAD
| OPTIONS } ]

```

soap-type-spec-string :

```

SOAP[: { RPC | DOC }

```

parameter :

```

parameter-mode parameter-name datatype [ DEFAULT expression ]

```

parameter-mode :

```

IN
| OUT
| INOUT

```

url-string :

```

{ HTTP: | HTTPS: | HTTPS_FIPS: }//[user:password@]hostname[:port][/path]

```

protocol-option-string

```

[ http-option-list ]
[ , soap-option-list ]

```

http-option-list :

```

HTTP(
[ CH[UNK]= { ON | OFF | AUTO } ]
[ ; VER[SION]= { 1.0 | 1.1 };kto=number-of-seconds ]
[ REDIR(COUNT=count, STATUS=status-list) ]
)

```

soap-option-list :

SOAP(OP[ERATION]=soap-operation-name)

Parameters

OR REPLACE clause Specifying CREATE OR REPLACE PROCEDURE creates a new procedure, or replaces an existing procedure with the same name. This clause changes the definition of the procedure, but preserves existing privileges. An error is returned if you attempt to replace a procedure that is already in use.

procedure-name The name of the procedure.

parameter Parameter names must conform to the rules for other database identifiers such as column names. They must be a valid SQL data type.

Only SOAP requests support the transmission of typed data such as FLOAT, INT, and so on. HTTP requests support the transmission of strings only, so you are limited to CHAR types.

When procedures are executed using the CALL statement, not all parameters must be specified. If a default value is provided in the CREATE PROCEDURE statement, missing parameters are assigned the default values. If an argument is not provided in the CALL statement, and no default is set, an error is given.

Parameters can be prefixed with one of the keywords IN, OUT, or INOUT. If you do not specify one of these values, parameters are INOUT by default. The keywords have the following meanings:

- **IN** The parameter is an expression that provides a value to the procedure.
- **OUT** The parameter is a variable that could be given a value by the procedure.
- **INOUT** The parameter is a variable that provides a value to the procedure, and could be given a new value by the procedure.

When defining a parameter, specify the %TYPE or %ROWTYPE attribute to set the data type(s) to the data type(s) of a column or row in a table, or view.

datatype The data type of the parameter. Set the data type explicitly, or specify the %TYPE or %ROWTYPE attribute to set the data type to the data type of another object in the database. Use %TYPE to set it to the data type of a column in a table or view. Use %ROWTYPE to set the data type to a composite data type derived from a row in a table or view.

RESULT clause The RESULT clause is required to use the procedure in a SELECT statement. The RESULT clause must return two columns. The first column contains HTTP response header, status, and response body attributes, while the second column contains the values for these attributes. The RESULT clause must specify two character data types. For example, VARCHAR or LONG VARCHAR. If the RESULT clause is not specified, the default column names are Attribute and Value and their data types are LONG VARCHAR.

URL clause Specifies the URI of the web service. The optional user name and password parameters provide a means of supplying the credentials needed for HTTP basic authentication. HTTP basic authentication base-64 encodes the user and password information and passes it in the Authentication header of the HTTP request. When specified in this way, the user name and password are passed unencrypted, as part of the URL.

For procedures of type HTTP:GET, query parameters can be specified within the URL clause in addition to being automatically generated from parameters passed to a procedure.

URL 'http://localhost/service?parm=1

Specifying HTTPS_FIPS forces the system to use the FIPS-certified libraries. If HTTPS_FIPS is specified, but no FIPS-certified libraries are present, libraries that are not FIPS-certified are used instead.

To use a certificate from the operating system certificate store, specify a URL beginning with **https://**.

TYPE clause Specifies the format used when making the web service request. SOAP:RPC is used when SOAP is specified or no TYPE clause is included. HTTP:POST is used when HTTP is specified.

The TYPE clause allows the specification of a MIME-type for HTTP:POST and HTTP:PUT types. When HTTP:PUT is used, then a MIME-type must be specified. The *MIME-type* specification is used to set the Content-Type request header and set the mode of operation to allow only a single call parameter to populate the body of the request. Only zero or one parameter may remain when making a web service stored procedure call after parameter substitutions have been processed. Calling a web service procedure with a NULL value or no parameter (after substitutions) results in a request with no body and a content-length of zero. When a MIME-type is specified then the single body parameter is sent in the request as is, so the application must ensure that the content is formatted to match the MIME-type.

Some typical MIME-types include:

- text/plain
- text/html
- text/xml

When no MIME-type is specified, parameter names and values (multiple parameters are permitted) are URL encoded within the body of the HTTP request.

The keywords for the TYPE clause have the following meanings:

- **'HTTP:GET'** By default, this type uses the application/x-www-form-urlencoded MIME-type for encoding parameters specified in the URL.

For example, the following request is produced when a client submits a request from the URL, <http://localhost/WebServiceName?arg1=param1&arg2=param2>:

```
GET /WebServiceName?arg1=param1&arg2=param2 HTTP/1.1
// <End of Request - NO BODY>
```

- **'HTTP:POST'** By default, this type uses the application/x-www-form-urlencoded MIME-type for encoding parameters specified in the body of a POST request. URL parameters are stored in the body of the request.

For example, the following request is produced when a client submits a request the URL, <http://localhost/WebServiceName?arg1=param1&arg2=param2>:

```
POST /WebServiceName HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 19
arg1=param1&arg2=param2
// <End of Request>
```

- **HTTP:PUT** HTTP:PUT is similar to HTTP:POST, but the HTTP:PUT type does not have a default media type.

The following example demonstrates how to configure a general purpose client procedure that uploads data to a database server running the %SQLANYSAMPI7%\SQLAnywhere\HTTP\put_data.sql sample:

```
CREATE OR REPLACE PROCEDURE CPUT([data] LONG VARCHAR, resnm LONG VARCHAR,
mediatype LONG VARCHAR)
  URL 'http://localhost/resource/!resnm'
  TYPE 'HTTP:PUT:!mediatype';

CALL CPUT('hello world', 'hello', 'text/plain');
```

- **HTTP:DELETE** A web service client procedure can be configured to delete a resource located on a server. Specifying the media type is optional.

The following example demonstrates how to configure a general purpose client procedure that deletes a resource from a database server running the put_data.sql sample:

```
CREATE OR REPLACE PROCEDURE CDEL(resnm LONG VARCHAR, mediatype LONG
VARCHAR)
  URL 'http://localhost/resource/!resnm'
  TYPE 'HTTP:DELETE:!mediatype';

CALL CDEL('hello', 'text/plain');
```

- **HTTP:HEAD** The HEAD method is identical to a GET method but the server does not return a body. A media type can be specified.

```
CREATE OR REPLACE PROCEDURE CHEAD(resnm LONG VARCHAR)
  URL 'http://localhost/resource/!resnm'
  TYPE 'HTTP:HEAD';

CALL CHEAD('hello');
```

- **HTTP:OPTIONS** The OPTIONS method is identical to a GET method but the server does not return a body. A media type can be specified. This method allows cross-origin resource sharing (CORS).
- **'SOAP:RPC'** This type sets the Content-Type header to 'text/xml'. SOAP operations and parameters are encapsulated in SOAP envelope XML documents.
- **'SOAP:DOC'** This type sets the Content-Type header to 'text/xml'. It is similar to the SOAP:RPC type but allows you to send richer data types. SOAP operations and parameters are encapsulated in SOAP envelope XML documents.

Specifying a MIME-type for the TYPE clause automatically sets the Content-Type header to that MIME-type.

HEADER clause When creating HTTP web service client procedures, use this clause to add, modify, or delete HTTP request header entries. The specification of headers closely resembles the format specified in RFC2616 Hypertext Transfer Protocol, HTTP/1.1, and RFC822 Standard for ARPA Internet Text Messages, including the fact that only printable ASCII characters can be specified for HTTP headers, and they are case-insensitive.

Headers can be defined as *header-name:value-name* pairs. Each header must be delimited from its value with a colon (:) and therefore cannot contain a colon. You can define multiple headers

by delimiting each pair with \n, \x0d\n, <LF> (line feed), or <CR><LF>. (carriage return followed by a line feed)

Multiple contiguous white spaces within the header are converted to a single white space.

CERTIFICATE clause To make a secure (HTTPS) request, a client must have access to the certificate used to sign the HTTP server's certificate (or any certificate higher in the signing chain). The necessary information is specified in a string of semicolon-separated keyword=value pairs. The following keywords are available:

Keyword	Abbreviation	Description
file		The file name of the certificate or specify * to use a certificate from the operating system certificate store. Cannot be specified if either the certificate or certificate_name keyword is specified.
certificate	cert	The certificate itself. Cannot be specified if either the file or certificate_name keyword is specified.
certificate_name	cert_name	The name of a certificate stored in the database. Cannot be specified if either the file or certificate keyword is specified.
company	co	The company specified in the certificate.
unit		The company unit specified in the certificate.
name		The common name specified in the certificate.
skip_certificate_name_check		Specify ON to prevent checking the database server certificate. <div data-bbox="1079 1434 1393 1818" style="border: 1px solid black; padding: 10px; margin-top: 10px;"> <p>Note</p> <p>Setting this option to ON is not recommended because this setting prevents the database server from fully authenticating the HTTP server.</p> </div>

Certificates are required only for requests that are either directed to an HTTPS server, or can be

redirected from a non-secure to a secure server. Only PEM formatted certificates are supported.

CLIENTPORT clause Identifies the port number on which the HTTP client procedure communicates using TCP/IP. It is provided for and recommended only for connections through firewalls that filter "outgoing" TCP/IP connections. You can specify a single port number, ranges of port numbers, or a combination of both; for example, `CLIENTPORT '85,90-97'`.

PROXY clause

Specifies the URI of a proxy server. For use when the client must access the network through a proxy. The *proxy-string* is usually an HTTP or HTTPS url-string. This is site specific information that you usually need to obtain from your network administrator. This clause indicates that the procedure is to connect to the proxy server and send the request to the web service through it. For an example, the following PROXY clause sets the proxy server to `proxy.example.com`:

```
PROXY http://proxy.example.com
```

SET clause Specifies protocol-specific behavior options for HTTP and SOAP. The following list describes the supported SET options. CHUNK, VERSION, REDIR and kto apply to the HTTP protocol, and OPERATION applies to the SOAP protocol.

- **'HTTP(CH[UNK]=option)'** (HTTP or SOAP) This option allows you to specify whether to use chunking. Chunking allows HTTP messages to be broken up into several parts. Possible values are ON (always chunk), OFF (never chunk), and AUTO (chunk only if the contents, excluding auto-generated markup, exceeds 8196 bytes). For example, the following SET clause enables chunking:

```
SET 'HTTP(CHUNK=ON)'
```

If the CHUNK option is not specified, the default behavior is AUTO. If a chunked request fails in AUTO mode with a status of 505 `HTTP Version Not Supported`, or with 501 `Not Implemented`, or with 411 `Length Required`, the client retries the request without chunked transfer-coding.

Set the CHUNK option to OFF (never chunk) if the HTTP server does not support chunked transfer-coded requests.

Since CHUNK mode is a transfer encoding supported starting in HTTP version 1.1, setting CHUNK to ON requires that the version (VER) be set to 1.1, or not be set at all, in which case 1.1 is used as the default version.

- **'HTTP(VER[SION]=ver;kto=number-of-seconds)'** (HTTP or SOAP) This option allows you to specify the version of HTTP protocol that is used for the format of the HTTP message. For example, the following SET clause sets the HTTP version to 1.1:

```
SET 'HTTP(VERSION=1.1)'
```

Possible values are 1.0 and 1.1. If VERSION is not specified:

- if CHUNK is set to ON, 1.1 is used as the HTTP version
- if CHUNK is set to OFF, 1.0 is used as the HTTP version
- if CHUNK is set to AUTO, either 1.0 or 1.1 is used, depending on whether the client is sending in CHUNK mode
- **kto=number-of-seconds** Specifies the keep-alive timeout criteria (kto), permitting a web client procedure to instantiate and cache a keep-alive HTTP/HTTPS connection for a period of time. To cache an HTTP keep-alive connection, the HTTP

version must be set to 1.1 and `kto` set to a non-zero value. `kto` may be useful, for HTTPS connections particularly, if you notice a significant performance difference between HTTP and HTTPS connections. A database connection can only cache a single keep-alive HTTP connection. Subsequent calls to a web client procedure using the same URI reuse the keep-alive connection. Therefore, the executing web client call must have a URI whose scheme, destination host and port match that of the cached URI, and the HEADER clause must not specify `Connection: close`. When `kto` is not specified, or is set to zero, HTTP/HTTPS connections are not cached.

- o **'REDIR(COUNT=*count*, STATUS=*status-list*)'** (HTTP or SOAP) The HTTP response status codes such as '302 Found' and '303 See Other' are used to redirect web applications to a new URI, particularly after an HTTP POST has been performed. For example, a client request could be:

```
GET /people/alice HTTP/1.1
Host: www.example.com
Accept: text/html, application/xhtml+xml
Accept-Language: en, de
```

The web server response could be:

```
HTTP/1.1 302 Found
Location: http://www.example.com/people/alice.en.html
```

In response, the client would send another HTTP request to the new URI. The REDIR option allows you to control the maximum number of redirections allowed and which HTTP response status codes to automatically redirect.

For example, `SET 'REDIR(count=3, status=301,307)'` allows a maximum limit of 3 re-directions and permits redirection for 301 and 307 statuses. If one of the other redirection status codes such as 302 or 303 is received, an error is issued (SQLCODE -983).

The default redirection limit *count* is 5. By default, an HTTP client procedure will automatically redirect in response to all HTTP redirection status codes (301, 302, 303, 307). To disallow all redirection status codes, use `SET REDIR(COUNT=0)`. In this mode, a redirection response does not result in an error (SQLCODE -983). Instead, a result set is returned with the HTTP status and response headers. This permits a caller to conditionally reissue the request based on the URI contained in the Location header.

A web service procedure specifying a POST HTTP method which receives a '303 See Other' status issues a redirect request using the GET HTTP method.

The Location header can contain either an absolute path or a relative path. The HTTP client procedure will handle either. The header can also include query parameters and these are forwarded to the redirected location. For example, if the header contained parameters such as the following, the subsequent GET or a POST will include these parameters.

```
Location: alternate_service?a=1&b=2
```

In the above example, the query parameters are `a=1&b=2`.

- o **'SOAP(OP[ERATION]=*soap-operation-name*)'** (SOAP only) This option allows you to specify the name of the SOAP operation, if it is different from the name of the procedure you are creating. The value of OPERATION is analogous to the name of a remote

procedure call. For example, if you wanted to create a procedure called `accounts_login` that calls a SOAP operation called `login`, you would specify something like the following:

```
CREATE PROCEDURE accounts_login(
    name LONG VARCHAR,
    pwd LONG VARCHAR )
SET 'SOAP(OPERATION=login)'
```

If the `OPERATION` option is not specified, the name of the SOAP operation must match the name of the procedure you are creating.

The following statement shows how several *protocol-option* settings are combined in the same `SET` clause:

```
CREATE PROCEDURE accounts_login(
    name LONG VARCHAR,
    pwd LONG VARCHAR )
SET 'HTTP ( CHUNK=ON; VERSION=1.1 ), SOAP( OPERATION=login )'
...
```

SOAPHEADER clause (SOAP format only) When declaring a SOAP web service as a procedure, use this clause to specify one or more SOAP request header entries. A SOAP header can be declared as a static constant, or can be dynamically set using the parameter substitution mechanism (declaring `IN`, `OUT`, or `INOUT` parameters for `hd1`, `hd2`, and so on). A web service procedure can define one or more `IN` mode substitution parameters, and a single `INOUT` or `OUT` substitution parameter.

The following example illustrates how a client can specify the sending of several header entries using parameter substitution and receiving the response SOAP header data:

```
CREATE PROCEDURE soap_client
(INOUT hd1 LONG VARCHAR, IN hd2 LONG VARCHAR, IN hd3 LONG VARCHAR)
URL 'localhost/some_endpoint'
SOAPHEADER '!hd1!hd2!hd3';
```

NAMESPACE clause (SOAP format only) This clause identifies the method namespace usually required for both `SOAP:RPC` and `SOAP:DOC` requests. The SOAP server handling the request uses this namespace to interpret the names of the entities in the SOAP request message body. The namespace can be obtained from the WSDL (Web Services Description Language) of the SOAP service available from the web service server. The default value is the procedure's URL, up to but not including the optional path component.

You can specify a variable name for *namespace-string*. If the variable is `NULL`, the namespace property is ignored.

Remarks

Parameter values are passed as part of the request. The syntax used depends on the type of request. For `HTTP:GET`, the parameters are passed as part of the URL; for `HTTP:POST` requests, the values are placed in the body of the request. Parameters to SOAP requests are always bundled in the request body.

You can create or replace a web services client procedure. You can use `PROC` as a synonym for `PROCEDURE`.

For SOAP requests, the procedure name is used as the SOAP operation name by default. For more information, see the SET clause.

Note You cannot create TEMPORARY web services procedures.

Note For *required* parameters that accept variable names, an error is returned if one of the following conditions is true:

- The variable does not exist
- The contents of the variable are NULL
- The variable exceeds the length allowed by the parameter
- The data type of the variable does not match that required by the parameter

Privileges

You must have the CREATE PROCEDURE system privilege to create procedures owned by you. You must have the CREATE ANY PROCEDURE or CREATE ANY OBJECT system privilege to create procedures owned by others.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.
- **Transact-SQL** Not supported by Adaptive Server Enterprise.

Example

1. The following example creates a web service client procedure named FtoC.

```
CREATE PROCEDURE FtoC( IN temperature FLOAT,  
    INOUT inoutheader LONG VARCHAR,  
    IN inheader LONG VARCHAR )  
URL 'http://localhost:8082/FtoCService'  
TYPE 'SOAP:DOC'  
SOAPHEADER '!inoutheader!inheader';
```

2. The following example creates a secure web service client procedure named SecureSendWithMimeType that uses a certificate stored in the database.

```
CREATE CERTIFICATE client_cert
FROM FILE 'C:\\Users\\Public\\Documents\\SQL Anywhere
17\\Samples\\Certificates\\rsaroot.crt';
```

```
CREATE PROCEDURE SecureSendWithMimeType(
    value LONG VARCHAR,
    mimeType LONG VARCHAR,
    urlSpec LONG VARCHAR
)
URL '!urlSpec'
CERTIFICATE 'certificate_name=client_cert'
TYPE 'HTTPS:POST:!mimeType';

CALL SecureSendWithMimeType('<hello>this is xml</hello>',
    'text/xml',
    'https://localhost:4043/EchoService'
);
```

3. The following example creates a procedure named SecureSendWithMimeType that uses a certificate from the operating system certificate store:

```
CREATE PROCEDURE SecureSendWithMimeType(
    value LONG VARCHAR,
    mimeType LONG VARCHAR,
    urlSpec LONG VARCHAR
)
URL '!urlSpec'
CERTIFICATE 'file=*'
TYPE 'HTTPS:POST:!mimeType';
```

4. The following example creates a procedure named SecureSendWithMimeType that verifies that the certificate myrootcert.crt is at the root of the database server's certificate's signing chain, but does no other checking:

```
CREATE PROCEDURE SecureSendWithMimeType(
    value LONG VARCHAR,
    mimeType LONG VARCHAR,
    urlSpec LONG VARCHAR
)
URL '!urlSpec'
CERTIFICATE 'file=myrootcert.crt;skip_certificate_name_check=ON'
TYPE 'HTTPS:POST:!mimeType';
```

5. The following example creates a procedure using a variable in the NAMESPACE clause
- The following statements create a variable for a NAMESPACE clause:

```
CREATE VARIABLE @ns LONG VARCHAR
SET @ns = 'http://wsdl.domain.com/';
```

- b. The following statement creates a procedure named FtoC that uses a variable in the NAMESPACE clause:

```
CREATE PROCEDURE FtoC( IN temperature FLOAT,  
    INOUT inoutheader LONG VARCHAR,  
    IN inheader LONG VARCHAR )  
URL 'http://localhost:8082/FtoCService'  
TYPE 'SOAP:DOC'  
SOAPHEADER '!inoutheader!inheader'  
NAMESPACE @ns;
```

Related Information

[SOAP structured data types](#)

The XML data type is supported for use as return values and parameters within web service functions and procedures.

[HTTP and SOAP request structures](#)

All parameters to a function or procedure, unless used during parameter substitution, are passed as part of the web service request. The format in which they are passed depends on the type of the web service request.

[HTTP request header management](#)

HTTP request headers can be added, changed, or removed with the HEADER clause of the CREATE PROCEDURE and CREATE FUNCTION statements.

[Variables supplied to web services](#)

Variables can be supplied to a web service in various ways depending on the web service type.

[The database server as an HTTP web server](#)

The database server contains a built-in HTTP web server that allows you to provide online web services from a database.

[Web client application development](#)

SQL Anywhere databases can act as web client applications to access web services hosted on a SQL Anywhere web server or on third party web servers.

[Substitution parameters used for clause values](#)

Declared parameters to a stored procedure or function are automatically substituted for placeholders within a clause definition each time the stored procedure or function is run.

[%TYPE and %ROWTYPE attributes](#)

In addition to explicitly setting the data type for an object, you can also set the data type by specifying the %TYPE and %ROWTYPE attributes.

[Tutorial: Using a database server to access a SOAP/DISH service](#)

This tutorial illustrates how to create a SOAP server that converts a web client-supplied Fahrenheit temperature value to Celsius.

[Tutorial: Create a web server and access it from a web client](#)

This tutorial illustrates how to create a web server and then send requests to it from a web client database server.

[ALTER PROCEDURE statement](#)

Modifies a procedure.

[CALL statement](#)

Invokes a procedure.

[CREATE FUNCTION statement](#)

Creates a user-defined SQL function in the database.

[CREATE FUNCTION statement \[Web service\]](#)

Creates a web client function that makes an HTTP or SOAP over HTTP request.

[CREATE PROCEDURE statement](#)

Creates a user-defined SQL procedure in the database.

[CREATE PROCEDURE statement \[T-SQL\]](#)

Creates a new procedure in the database in a manner compatible with Adaptive Server Enterprise.

[CREATE PROCEDURE statement \[External call\]](#)

Creates an interface to a native or external procedure.

[DROP PROCEDURE statement](#)

Removes a procedure from the database.

[GRANT statement](#)

Grant system and object-level privileges to users and roles.

[remote_idle_timeout option](#)

Controls how many seconds of inactivity web service client procedures and functions tolerate.

[ClientPort \(CPORT\) protocol option \(client side only\)](#)

Designates the port number on which the client application communicates using TCP/IP.

[HTTP request failed. Status code '%1'](#)

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

CREATE PROCEDURE statement [T-SQL]

Creates a new procedure in the database in a manner compatible with Adaptive Server Enterprise.

Syntax

The following subset of the Transact-SQL CREATE PROCEDURE statement is supported in SQL Anywhere.

```
CREATE [ OR REPLACE ]PROCEDURE [owner.]procedure-name
[ NO RESULT SET ]
[ [ ( ] @parameter-name data-type [ = default ] [ OUTPUT ], ... [ ) ] ]
[ WITH RECOMPILE ] AS statement-list
```

Parameters

OR REPLACE clause Specifying CREATE OR REPLACE PROCEDURE creates a new procedure, or replaces an existing procedure with the same name. This clause changes the definition of the procedure, but preserves existing privileges. An error is returned if you attempt to replace a procedure that is already in use.

NO RESULT SET clause Declares that no result set is returned by this procedure. This is useful when an external environment needs to know that a procedure does not return a result set.

WITH RECOMPILE clause This clause is accepted for Transact-SQL compatibility, but is ignored. SQL Anywhere always recompiles procedures the first time they are executed after a database is started, and stores the compiled procedure until the database is stopped.

Remarks

The following differences between Transact-SQL and SQL Anywhere statements (Watcom SQL) are listed to help those writing in both dialects:

- **Variable names prefixed by @** The @ sign denotes a Transact-SQL variable name, while Watcom SQL variables can be any valid identifier, and the @ prefix is optional.
- **Input and output parameters** Watcom SQL procedure parameters are INOUT by default or can be specified as IN, OUT, or INOUT. Transact-SQL procedure parameters are INPUT parameters by default. They can be specified as input/output with the addition of the OUTPUT keyword. There are no output-only parameters in the Transact-SQL dialect.

When you use the Watcom SQL dialect to declare a parameter OUT, it is output-only. The mixing of dialects is not recommended because it can cause problems when the procedure declaration is unloaded and used to rebuild the database. If the procedure declaration is unloaded and used to rebuild the database, the rebuilt procedure declaration is in the Transact-SQL dialect, the OUTPUT keyword is used, and the parameter is input/output.

- **Parameter default values** Watcom SQL procedure parameters are given a default value using the keyword DEFAULT, while Transact-SQL uses an equality sign (=) to provide the default value.
- **Returning result sets** Watcom SQL uses a RESULT clause to specify returned result sets. In Transact-SQL procedures, the column names or alias names of the first query are returned to the calling environment.

The following Transact-SQL procedure illustrates how result sets are returned from Transact-SQL stored procedures:

```
CREATE PROCEDURE showdept @deptname varchar(30)
AS
    SELECT Employees.Surname, Employees.GivenName
    FROM Departments, Employees
    WHERE Departments.DepartmentName = @deptname
    AND Departments.DepartmentID = Employees.DepartmentID;
```

The following is the corresponding Watcom SQL procedure:

```
CREATE PROCEDURE showdept2(in deptname
    varchar(30) )
RESULT ( lastname char(20), firstname char(20))
ON EXCEPTION RESUME
BEGIN
    SELECT Employees.Surname, Employees.GivenName
    FROM Departments, Employees
    WHERE Departments.DepartmentName = deptname
    AND Departments.DepartmentID = Employees.DepartmentID
END;
```

- **Procedure body** The body of a Transact-SQL procedure is a list of Transact-SQL statements prefixed by the AS keyword. The body of a Watcom SQL procedure is a compound statement, bracketed by BEGIN and END keywords.

Privileges

You must have the CREATE PROCEDURE privilege to create procedures owned by you. You must have the CREATE ANY PROCEDURE or CREATE ANY OBJECT privilege to create procedures owned by others.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.
- **Transact-SQL** SQL Anywhere supports a subset of the Adaptive Server Enterprise CREATE PROCEDURE statement syntax.

Only Transact-SQL SQL procedures are supported in the SQL Anywhere Transact-SQL dialect. To create an external procedure you must use Watcom SQL syntax. Adaptive Server Enterprise does not support the NO RESULT SET clause. If the Transact-SQL WITH RECOMPILE optional clause is supplied, it is ignored. SQL Anywhere always recompiles procedures the first time they are executed after a database is started, and stores the compiled procedure until the database is stopped.

Groups of Transact-SQL procedures are not supported in SQL Anywhere.

Related Information

[CREATE FUNCTION statement](#)

Creates a user-defined SQL function in the database.

[CREATE PROCEDURE statement](#)

Creates a user-defined SQL procedure in the database.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

CREATE PROCEDURE statement

Creates a user-defined SQL procedure in the database.

Syntax

```
CREATE [ OR REPLACE | TEMPORARY ] PROCEDURE [ owner.] procedure-name
( [ parameter, ... ] )
[ RESULT ( result-column, ... ) | NO RESULT SET ]
[ SQL SECURITY { INVOKER | DEFINER } ]
[ ON EXCEPTION RESUME ]
compound-statement | AT location-string
```

parameter :

parameter-mode *parameter-name* *data-type* [**DEFAULT** *expression*]

| **SQLCODE**

| **SQLSTATE**

parameter-mode :

IN

| **OUT**

| **INOUT**

result-column : *column-name* *data-type*

Parameters

OR REPLACE clause Specifying CREATE OR REPLACE PROCEDURE creates a new procedure, or replaces an existing procedure with the same name. This clause changes the definition of the procedure, but preserves existing privileges. An error is returned if you attempt to replace a procedure that is already in use.

TEMPORARY clause Specifying CREATE TEMPORARY PROCEDURE means that the stored procedure is visible only by the connection that created it, and that it is automatically dropped when the connection is dropped. Temporary stored procedures can also be explicitly dropped. You cannot perform ALTER, GRANT, or REVOKE on them, and, unlike other stored procedures, temporary stored procedures are not recorded in the catalog or transaction log.

Temporary procedures execute with the privileges of their creator (current user), or specified owner. You can specify an owner for a temporary procedure when:

- the temporary procedure is created within a permanent stored procedure
- the owner of the temporary and permanent procedure is the same

To drop the owner of a temporary procedure, you must drop the temporary procedure first.

Temporary stored procedures can be created and dropped when connected to a read-only database, and they cannot be external procedures.

For example, the following temporary procedure drops the fictitious table called CustRank. For this example, the procedure assumes that the table name is unique and can be referenced by the procedure creator without specifying the table owner:

```

CREATE TEMPORARY PROCEDURE drop_table( IN @TableName char(128) )
BEGIN
    IF EXISTS ( SELECT * FROM SYS.SYSTAB WHERE table_name = @TableName ) THEN
        EXECUTE IMMEDIATE 'DROP TABLE ' || @TableName || '';
        MESSAGE 'Table ' || @TableName || ' dropped' to client;
    END IF;
END;
CALL drop_table( 'CustRank' );

```

parameter Parameter names must conform to the rules for other database identifiers such as column names. They must be a valid SQL data type.

Parameters can be prefixed with one of the keywords IN, OUT, or INOUT. If you do not specify one of these values, parameters are INOUT by default. The keywords have the following meanings:

- **IN** The parameter is an expression that provides a value to the procedure.
- **OUT** The parameter is a variable that could be given a value by the procedure.
- **INOUT** The parameter is a variable that provides a value to the procedure, and could be given a new value by the procedure.

Set the data type explicitly, or specify the %TYPE or %ROWTYPE attribute to set the data type to the data type of another object in the database. Use %TYPE to set it to the data type of a column in a table or view. Use %ROWTYPE to set the data type to a composite data type derived from a row in a table or view. However, defining the data type using a %ROWTYPE that is set to a table reference variable (`TABLE REF (table-reference-variable) %ROWTYPE`) is not allowed.

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 an argument is not provided in the CALL statement, and no default is set, an error is given.

SQLSTATE and SQLCODE are special OUT parameters that output the SQLSTATE or SQLCODE value when the procedure ends. The SQLSTATE and SQLCODE special values can be checked immediately after a procedure call to test the return status of the procedure.

The SQLSTATE and SQLCODE special 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.

Specifying CREATE OR REPLACE PROCEDURE creates a new procedure, or replaces an existing procedure with the same name. This clause changes the definition of the procedure, but preserves existing privileges. You cannot use the OR REPLACE clause with temporary procedures. An error is returned if the procedure being replaced is already in use. Open cursors for a connection are closed when a CREATE OR REPLACE PROCEDURE statement is executed.

RESULT clause The RESULT clause declares the number and type of columns in the result set. The parenthesized list following the RESULT keyword defines the result column names and types. This information is returned by the Embedded SQL DESCRIBE or by ODBC SQLDescribeCol when a CALL statement is being described.

You can define the data type of the columns in the result set using the %TYPE or %ROWTYPE attribute. Use %TYPE to set a column type to the data type of a column in a table or view. Use %ROWTYPE to set the data type to a composite data type derived from a row in a table or view.

If a RESULT clause is specified, it must be the first clause of the statement.

Some procedures can produce more than one result set, 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 GivenName
        FROM GROUP0.Employees
    ELSE
        SELECT Surname, GivenName
        FROM GROUP0.Employees
    END IF
END;
```

Procedures with variable result sets 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, to get the proper shape of result set. The CURSOR *cursor-name* clause on the DESCRIBE statement is required.
- **ODBC, OLE DB, ADO.NET** Variable result-set procedures can be used by applications using these interfaces. The proper description of the result sets is carried out by the driver or provider.
- **Open Client applications** Variable result-set procedures can be used by Open Client applications.
- **Web services** Web services rely on the RESULTS clause of the stored procedure to determine the number and types of the column in the result set. Web services do not support procedures that return multiple result sets, nor do they support variable result sets through the use of EXECUTE IMMEDIATE.

Note If an EXECUTE IMMEDIATE statement that includes a WITH RESULT SET ON clause is used in the procedure, and if the result set that is returned from the statement is the same as the result set that is returned from the procedure, then only the first column of the EXECUTE IMMEDIATE statement's result set is returned.

If your procedure returns only one result set, 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.

To handle multiple result sets, ODBC must describe the currently executing cursor, not the procedure's defined result set. Therefore, ODBC does not always describe column names as defined in the RESULT clause of the procedure definition. To avoid this problem, use column aliases in the SELECT statement that generates the result set.

NO RESULT SET clause If a NO RESULT SET clause is specified, it must be the first clause of the statement.

Declares that no result set is returned by this procedure. This is useful when an external environment needs to know that a procedure does not return a result set.

SQL SECURITY clause The SQL SECURITY clause defines whether the procedure is executed as the INVOKER (the user who is calling the procedure), or as the DEFINER (the user

who owns the procedure). The default is DEFINER.

When SQL SECURITY INVOKER is specified, more memory is used because annotation must be done for each user that calls the procedure. When SQL SECURITY INVOKER is specified, name resolution is done as the invoker as well. Therefore, make sure to qualify all object names (tables, procedures, and so on) with their appropriate owner. For example, suppose user1 creates the following procedure:

```
CREATE PROCEDURE user1.myProcedure()  
  RESULT( columnA INT )  
  SQL SECURITY INVOKER  
BEGIN  
  SELECT columnA FROM table1;  
END;
```

If user2 attempts to run this procedure and a table user2.table1 *does not* exist, a table lookup error results. Additionally, if a user2.table1 *does* exist, that table is used instead of the intended user1.table1. To prevent this situation, qualify the table reference in the statement (user1.table1, instead of just table1).

ON EXCEPTION RESUME clause This clause enables Transact-SQL-like error handling to be used within a Watcom SQL syntax procedure.

If you use ON EXCEPTION RESUME, the procedure takes an action that depends on the setting of the on_tsq_error option. If on_tsq_error is set to Conditional (the default) the execution continues if the next statement handles the error; otherwise, it exits.

Error-handling statements include the following:

- IF
- SELECT @variable =
- CASE
- LOOP
- LEAVE
- CONTINUE
- CALL
- EXECUTE
- SIGNAL
- RESIGNAL
- DECLARE
- SET VARIABLE

You should not use explicit error handling code with an ON EXCEPTION RESUME clause.

This clause is ignored within the TRY block of a BEGIN...END statement.

compound-statement A set of SQL statements bracketed by BEGIN and END, and separated by semicolons.

AT clause Create a proxy stored procedure on the current database for a remote procedure specified by *location-string*. The AT clause supports the semicolon (;) as a field delimiter in *location-string*. If no semicolon is present, a period is the field delimiter. This allows file names

and extensions to be used in the database and owner fields.

The string in the AT clause can also contain local or global variable names enclosed in braces ({*variable-name*}). The SQL variable name must be of type CHAR, VARCHAR, or LONG VARCHAR. For example, an AT clause that contains '*bostonase.master.dbo.{@myprocedure}*' indicates that *@myprocedure* is a SQL variable and that the current contents of the *@myprocedure* variable should be substituted when the remote procedure is used.

If a remote procedure can return a result set, even if it does not always return one, then the local procedure definition must contain a RESULT clause.

Remarks

The CREATE PROCEDURE statement creates a procedure in the database. A procedure is invoked with a CALL statement.

You can create permanent stored procedures that call external or native procedures written in a variety of programming languages.

You can use PROC as a synonym for PROCEDURE.

When referencing a temporary table from multiple procedures, a potential issue can arise if the temporary table definitions are inconsistent and statements referencing the table are cached.

The body of a procedure is a compound statement. The compound statement starts with a BEGIN statement and concludes with an END statement. For NewDepartment the compound statement is a single INSERT bracketed by BEGIN and END statements.

Parameters to procedures can be marked as one of IN, OUT, or INOUT. By default, parameters are INOUT parameters. All parameters to the NewDepartment procedure are IN parameters, as they are not changed by the procedure. You should set parameters to IN if they are not used to return values to the caller.

Privileges

You must have the CREATE PROCEDURE system privilege to create procedures owned by you. You must have the CREATE ANY PROCEDURE or CREATE ANY OBJECT privilege to create procedures owned by others. To create external procedures, you must also have the CREATE EXTERNAL REFERENCE system privilege.

You do not need any privilege to create temporary procedures.

To replace an existing procedure, you must own the procedure or have one of the following:

- CREATE ANY PROCEDURE and DROP ANY PROCEDURE system privileges.
- CREATE ANY OBJECT and DROP ANY OBJECT system privileges.
- ALTER ANY OBJECT or ALTER ANY PROCEDURE system privileges.

Side effects

Automatic commit, even for temporary procedures.

Standards

- **ANSI/ISO SQL Standard** CREATE PROCEDURE is a core feature of the ANSI/ISO SQL Standard, but some of its components supported in SQL Anywhere are optional SQL language features. A subset of these features includes:
 - The SQL SECURITY clause is optional ANSI/ISO SQL Language Feature T324.

- The ability to pass a LONG VARCHAR, LONG NVARCHAR, or LONG BINARY value to a SQL procedure is ANSI/ISO SQL Language Feature T041.
- The ability to create or modify a schema object within a SQL procedure, using statements such as CREATE TABLE or DROP TRIGGER, is ANSI/ISO SQL Language Feature T651.
- The ability to use a dynamic-SQL statement within a SQL procedure, including statements such as EXECUTE IMMEDIATE, PREPARE, and DESCRIBE, is ANSI/ISO SQL Language Feature T652.

Several clauses of the CREATE PROCEDURE statement are not in the standard. These include:

- The TEMPORARY clause.
- The ON EXCEPTION RESUME clause.
- The AT clause.
- The optional DEFAULT clause for a specific routine parameter.
- The RESULT and NO RESULT SET clauses. The ANSI/ISO SQL Standard uses the RETURNS keyword.
- The optional OR REPLACE clause.

- **Transact-SQL** CREATE PROCEDURE is supported by Adaptive Server Enterprise.

Example

The following procedure queries the Employees table and returns salaries that are within the specified percent (*percentage*) of a specified salary (*sal*):


```
CREATE OR REPLACE PROCEDURE AverageEmployees( IN percentage NUMERIC( 5,3), IN sal
NUMERIC( 20, 3 ) )
RESULT( Department CHAR(40), GivenName person_name_t, Surname person_name_t,
Salary NUMERIC( 20, 3) )
BEGIN
    DECLARE maxS NUMERIC( 20, 3 );
    DECLARE minS NUMERIC( 20, 3 );

    IF percentage >= 1 THEN
        SET percentage = percentage / 100;
    ELSEIF percentage < 0 THEN
        SELECT 'Percentage error', 'Err','Err', -1;
        RETURN;
    END IF;

    SELECT MIN( E.Salary ), MAX( E.Salary ) INTO minS, maxS
    FROM GROUP0.Employees E;

    IF sal < minS OR sal > maxS THEN
        SELECT 'Salary out of bounds', 'Err', 'Err', -2;
        RETURN;
    END IF;

    SELECT D.DepartmentName, E.GivenName, E.Surname, E.Salary
    FROM GROUP0.Employees E JOIN Departments D ON E.DepartmentID = D.DepartmentID
    WHERE E.Salary BETWEEN sal *( 1 - percentage ) AND sal * ( 1 + percentage );
END;
```

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 GROUP0.Products
  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;
```

The following example replaces the ProductType procedure created in the previous example. After replacing the procedure, the parameters for Tee Shirt and Sweatshirt are updated:

```
CREATE OR REPLACE PROCEDURE ProductType (IN product_ID INT, OUT type CHAR(10))
BEGIN
  DECLARE prod_name CHAR(20);
  SELECT name INTO prod_name FROM GROUP0.Products
  WHERE ID = product_ID;
  CASE prod_name
  WHEN 'Tee Shirt' THEN
    SET type = 'T Shirt'
  WHEN 'Sweatshirt' THEN
    SET type = 'Long Sleeve 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 CompanyName,
           CAST(SUM(SalesOrderItems.Quantity *
                   Products.UnitPrice) AS INTEGER) VALUE
    FROM GROUPO.Customers
    LEFT OUTER JOIN SalesOrders
    LEFT OUTER JOIN SalesOrderItems
    LEFT OUTER JOIN Products
    GROUP BY CompanyName;
  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;

```

The following example creates the procedure NewDepartment, which performs an INSERT into the Departments table of the SQL Anywhere sample database, creating a new department:

```

CREATE PROCEDURE NewDepartment(
  IN id INT,
  IN name CHAR(35),
  IN head_id INT )
BEGIN
  INSERT
  INTO GROUPO.Departments ( DepartmentID,
                           DepartmentName, DepartmentHeadID )
  VALUES ( id, name, head_id );
END;

```

The following statement creates a procedure, DepartmentsCloseToCustomerLocation, and sets its IN parameter to the data type of the ID column in the Customers table using a %TYPE attribute:

```
CREATE OR REPLACE PROCEDURE DepartmentsCloseToCustomerLocation( IN customer_ID
Customers.ID%TYPE )
BEGIN
    DECLARE cust_rec Customers%ROWTYPE;

    SELECT City, State, Country
    INTO cust_rec.City, cust_rec.State, cust_rec.Country
    FROM Customers
    WHERE ID = customer_ID;

    SELECT Employees.Surname, Employees.GivenName, Departments.DepartmentName
    FROM Employees JOIN Departments
    ON Departments.DepartmentHeadID = Employees.EmployeeID
    WHERE Employees.City = cust_rec.City
    AND Employees.State = cust_rec.State
    AND Employees.Country = cust_rec.Country;
END;

CALL DepartmentsCloseToCustomerLocation(158);
```

Related Information

[EXECUTE IMMEDIATE used in procedures, triggers, user-defined functions, and batches](#)

The EXECUTE IMMEDIATE statement allows statements to be constructed using a combination of literal strings (in quotes) and variables.

[Result sets](#)

Procedures can return results in the form of a single row of data, or multiple rows.

[References to temporary tables within procedures](#)

Sharing a temporary table between procedures can cause problems if the table definitions are inconsistent.

[Stored procedures, triggers, batches, and user-defined functions](#)

Procedures and triggers store procedural SQL statements in a database.

[Remote procedure calls \(RPCs\)](#)

You can issue procedure calls to the remote servers.

[%TYPE and %ROWTYPE attributes](#)

In addition to explicitly setting the data type for an object, you can also set the data type by specifying the %TYPE and %ROWTYPE attributes.

[Creating remote procedures \(SQL Central\)](#)

Create a remote procedure.

[ALTER PROCEDURE statement](#)

Modifies a procedure.

[BEGIN statement](#)

Specifies a compound statement.

[CALL statement](#)

Invokes a procedure.

[CREATE FUNCTION statement](#)

Creates a user-defined SQL function in the database.

[CREATE PROCEDURE statement \[External call\]](#)

Creates an interface to a native or external procedure.

[CREATE PROCEDURE statement \[Web service\]](#)

Creates a user-defined web client procedure that makes HTTP or SOAP requests to an HTTP server.

[CREATE PROCEDURE statement \[T-SQL\]](#)

Creates a new procedure in the database in a manner compatible with Adaptive Server Enterprise.

[CREATE SERVER statement](#)

Creates a remote server or a directory access server.

[DROP PROCEDURE statement](#)

Removes a procedure from the database.

[EXECUTE IMMEDIATE statement \[SP\]](#)

Enables dynamically constructed statements to be executed from within a procedure.

[GRANT statement](#)

Grant system and object-level privileges to users and roles.

[SQL data types](#)

There are many SQL data types supported by the software.

[on_tsq1_error option](#)

Controls error-handling in stored procedures.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

CREATE PUBLICATION statement [MobiLink] [SQL Remote]

Creates a publication. In MobiLink, a publication identifies synchronized data in a SQL Anywhere remote database. In SQL Remote, publications identify replicated data in both consolidated and remote databases.

Syntax

- **MobiLink general use**

```
CREATE PUBLICATION [ IF NOT EXISTS ] [ owner. ] publication-name
( article-definition, ... )
```

article-definition :

```
TABLE table-name [ ( column-name, ... ) ]
[ WHERE search-condition ]
```

- **MobiLink scripted upload**

```
CREATE PUBLICATION [ IF NOT EXISTS ] [ owner. ] publication-name
WITH SCRIPTED UPLOAD
( article-definition, ... )
```

article-definition :

```
TABLE table-name [ ( column-name, ... ) ]
[ USING ( [ PROCEDURE ] [ owner. ] procedure-name
FOR UPLOAD { INSERT | DELETE | UPDATE }, ... ) ]
```

- **MobiLink download-only publications**

```
CREATE PUBLICATION [ IF NOT EXISTS ] [ owner. ] publication-name
FOR DOWNLOAD ONLY
( article-definition, ... )
```

article-definition : **TABLE** *table-name* [(*column-name*, ...)]

- **SQL Remote**

```
CREATE PUBLICATION [ IF NOT EXISTS ] [ owner. ] publication-name
( article-definition, ... )
```

article-definition :

```
TABLE table-name [ ( column-name, ... ) ]
[ WHERE search-condition ]
[ SUBSCRIBE BY expression ]
```

Parameters

IF NOT EXISTS clause When the IF NOT EXISTS clause is specified and the named publication already exists, no changes are made and an error is not returned.

article-definition Publications are built from articles. Each article identifies the rows and columns of a single table that are included in the publication. A publication may not contain two articles that refer to the same table.

If a list of column-names is included in an article, only those columns are included in the publication. If no column-names are listed, all columns in the table are included in the publication. For MobiLink synchronization, if column-names are listed then all columns in the primary key of the table must be included in the list.

In the MobiLink scripted upload syntax, which is used for publications that perform scripted uploads, the article description also registers the scripts that are used to define the upload.

In the MobiLink download-only publications, which is used for download-only publications, the article specifies only the tables and columns to be downloaded.

WHERE clause The WHERE clause lets you define the subset of rows in a table to be included in an article.

In MobiLink applications, the WHERE clause affects the rows included in the upload. (The download is defined by the download_cursor script.) In MobiLink SQL Anywhere remote databases, the WHERE clause can only refer to columns included in the article, and cannot contain subqueries, variables, or non-deterministic functions.

SUBSCRIBE BY clause In SQL Remote, 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.

Remarks

The CREATE PUBLICATION statement creates a publication in the database. A publication can be created for another user by specifying an owner name.

In MobiLink, publications are required in SQL Anywhere remote databases, and are optional in UltraLite databases. These publications and the subscriptions to them determine which data is uploaded to the MobiLink server.

You set options for a MobiLink publication with the ADD OPTION clause in the CREATE SYNCHRONIZATION SUBSCRIPTION statement or ALTER SYNCHRONIZATION SUBSCRIPTION statement.

The MobiLink scripted upload syntax creates a publication for scripted uploads. Use the USING clause to register the stored procedures that you want to use to define the upload. For each table, you can use up to three stored procedures: one each for inserts, deletes, and updates.

The MobiLink download-only publications syntax creates a download-only publication that can be synchronized with no transaction log file. When download-only publications are synchronized, downloaded rows may overwrite changes that were made to those rows in the remote database.

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 SQL Remote Extraction utility from a consolidated database automatically executes the correct CREATE PUBLICATION statement in the remote database.

For all syntaxes, you must have exclusive access to all tables referred to in the statement to execute the statement.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statement publishes all columns and rows of two tables.

```
CREATE PUBLICATION pub_contact (  
    TABLE GROUP0.Contacts,  
    TABLE GROUP0.Customers  
);
```

The following statement publishes only some columns of one table.

```
CREATE PUBLICATION pub_customer (  
    TABLE GROUP0.Customers ( ID, CompanyName, City )  
);
```

The following statement publishes only the rows for customer located in New York (NY) by including a WHERE clause that tests the State column of the Customers table.

```
CREATE PUBLICATION pub_customer (  
    TABLE GROUP0.Customers ( ID, CompanyName, City, State, Status )  
    WHERE State = 'NY'  
);
```

The following statement publishes only some rows by providing a subscribe-by value. This method can be used only with SQL Remote.

```
CREATE PUBLICATION pub_customer (  
    TABLE GROUP0.Customers ( ID, CompanyName, City, State )  
    SUBSCRIBE BY State  
);
```

The subscribe-by value is used as follows when you create a SQL Remote subscription.

```
CREATE SUBSCRIPTION TO pub_customer ( 'NY' )  
    FOR jsmith;
```

The following example creates a MobiLink publication that uses scripted uploads:


```

CREATE PUBLICATION pub WITH SCRIPTED UPLOAD (
  GROUP0.TABLE t1 (a, b, c) USING (
    PROCEDURE my.t1_ui FOR UPLOAD INSERT,
    PROCEDURE my.t1_ud FOR UPLOAD DELETE,
    PROCEDURE my.t1_uu FOR UPLOAD UPDATE
  ),
  GROUP0.TABLE t2 AS my_t2 USING (
    PROCEDURE my.t2_ui FOR UPLOAD INSERT
  )
);

```

The following example creates a download-only publication:

```

CREATE PUBLICATION p1 FOR DOWNLOAD ONLY (
  GROUP0.TABLE t1
);

```

Related Information

[Publications](#)

A publication is a database object that identifies the data that is to be synchronized. It defines the data to be uploaded, and it limits the tables that can be downloaded to. (The download is defined in the `download_cursor` script.)

[Publications and articles](#)

A **publication** defines the set of data to be replicated. A publication can include data from several database tables.

[Scripted upload](#)

Scripted upload applies only to MobiLink applications that use SQL Anywhere remote databases.

[Download-only publications](#)

You can create a publication that only downloads data to remote databases, and never uploads data. Download-only publications do not use a transaction log on the client.

[Publications for scripted upload](#)

To create a scripted upload publication, use the keywords `WITH SCRIPTED UPLOAD` and specify the stored procedures in the `USING` clause.

[Publishing only some rows in a table \(SQL Central\)](#)

To create a publication that contains only some of the rows in a table, you must write a search condition that matches only the rows you want to publish.

[Creating publications \(SQL Central\)](#)

Create publications based on existing tables in the consolidated database that consist of all the columns and rows in a table.

[Publishing only some columns in a table \(SQL Central\)](#)

Create a publication that contains all of the rows, but only some of the columns, of a table.

[Publishing only some rows using the SUBSCRIBE BY clause \(SQL Central\)](#)

Create a publication using the `SUBSCRIBE BY` clause.

[Publishing only some rows using a WHERE clause \(SQL Central\)](#)

Create a publication that uses a `WHERE` clause to include all the columns, but only some of the rows of a table.

[Replicating the primary key pool \(SQL\)](#)

Create a separate publication for the primary key pool and subscribe users to it.

[ALTER PUBLICATION statement \[MobiLink\] \[SQL Remote\]](#)

Alters a publication. In MobiLink, a publication identifies synchronized data in a SQL Anywhere remote database. In SQL Remote, a publication identifies replicated data in both consolidated and remote databases.

[DROP PUBLICATION statement \[MobiLink\] \[SQL Remote\]](#)

Drops a publication.

[CREATE SYNCHRONIZATION SUBSCRIPTION statement \[MobiLink\]](#)

Creates a subscription in a SQL Anywhere remote database between a MobiLink user and a publication.

[ALTER SYNCHRONIZATION SUBSCRIPTION statement \[MobiLink\]](#)

Alters the properties of a synchronization subscription in a SQL Anywhere remote database.

[SYSSYNC system view](#)

The SYSSYNC system view contains information relating to synchronization.

[Overlap partitions](#)

Data partitioning overlaps when the remote databases share data. For example, sales representatives can share customers.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CREATE REMOTE [MESSAGE] TYPE statement [SQL Remote]

Identifies a message link and return address for outgoing messages from a database.

Syntax

```
CREATE REMOTE [MESSAGE] TYPE message-system  
[ ADDRESS address-string ]
```

message-system :

FILE

| **FTP**

| **HTTP**

| **SMTP**

Parameters

message-system One of the message systems supported by SQL Remote. It must be one of the following values: FILE, FTP, or SMTP.

address-string A string containing a valid address for the specified message system.

Remarks

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.

The address is the publisher's address under the specified message system. If it is an email system, the address string must be a valid email 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.

To remove the address, execute a CREATE REMOTE MESSAGE TYPE statement without an ADDRESS clause.

The dbinit 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.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

When remote databases are extracted using the extraction utility, the following statement sets all recipients of file message-system messages to send messages back to the *company* subdirectory.

The statement also instructs dbremote to look in the *company* subdirectory for incoming messages.

```
CREATE REMOTE MESSAGE TYPE file  
ADDRESS 'company';
```

Related Information

[SQL Remote message systems](#)

In SQL Remote replication, a message system is a protocol for exchanging messages between the consolidated database and a remote database. SQL Remote exchanges data among databases using one or more underlying message systems.

[The FILE message system](#)

SQL Remote can be used even if you do not have an email or FTP system in place, by using the FILE message system.

[The FTP message system](#)

In the FTP message system, messages are stored in directories under the root directory of an FTP host. The host and root directory are specified by message system control parameters in the registry or initialization file, and the address of each user is the subdirectory where their messages are held.

[The SMTP message system](#)

With the SMTP system, SQL Remote sends messages using Internet mail.

[Creating a message type \(SQL Central\)](#)

Add message types for SQL Remote users.

[GRANT PUBLISH statement \[SQL Remote\]](#)

Grants publisher rights to a user ID. You must have the SYS_REPLICATION_ADMIN_ROLE system role to grant publisher rights.

[GRANT REMOTE statement \[SQL Remote\]](#)

Identifies a database immediately below the current database in a SQL Remote hierarchy, that will receive messages from the current database. These are called remote users.

[GRANT CONSOLIDATE statement \[SQL Remote\]](#)

Identifies the database immediately above the current database in a SQL Remote hierarchy, that will receive messages from the current database.

[DROP REMOTE MESSAGE TYPE statement \[SQL Remote\]](#)

Deletes a message type definition from a database.

[ALTER REMOTE MESSAGE TYPE statement \[SQL Remote\]](#)

Changes the publisher's message system, or the publisher's address for a given message system, for a message type that has been created.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

CREATE ROLE statement

Creates or replaces a role, creates a user-extended role, or modifies administrators for a role.

Syntax

```
CREATE [ OR REPLACE ] ROLE { role-name | FOR USER userid }  
[ WITH ADMIN [ ONLY ] administrator-userid [,...] ]
```

Parameters

- **OR REPLACE clause** Use this clause to create the role if it does not already exist or replace its administrators if it does exist.
- ***role-name*** Use this variable to assign a name to the new role. This name must be unique across all users and roles in the database.
- **FOR USER *userid* clause** Use this clause to convert the specified user into a user-extended role that can be assigned to others. The user must not already be extended as another role.
- **WITH ADMIN and WITH ADMIN ONLY *administrator-userid* clause** Optionally specify administrators for the role. WITH ADMIN means that *administrator-userid* can exercise the role and administer it. WITH ADMIN ONLY means that *administrator-userid* can only administer the role. If no clause is specified, then any user with the MANAGE ROLES system privilege can administer the role.

The `min_role_admins` database option controls the minimum number of administrators required for each role. If you do not specify enough administrators when creating the role, the statement returns an error.

Remarks

The name of the new role must not begin and end with 'SYS_' and '_ROLE', respectively. For example SYS_MyBackup_ROLE is not an acceptable name for a user-defined role, whereas MyBackup_ROLE and SYS_MyBackup are acceptable.

If an ADMIN clause is specified, only the specified users can administer the roles. If no ADMIN clause is specified, then by default the role is granted to the MANAGE ROLES system privilege, with administrative rights only. This means that global administrators can administer the role.

To create a user-extended role (that is, to extend a user to be a role), use the [CREATE ROLE FOR USER *userid*](#) syntax.

Use the GRANT statements to grant system privileges to the role.

Privileges

You must have the MANAGE ROLES system privilege to create a new role.

If the OR REPLACE clause is specified and the role already exists, you must also have administrative rights over the role.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statement creates the Sales role. Any user with the MANAGE ROLES system privilege can administer the role.

```
CREATE ROLE Sales;
```

The following statement extends user JaneSmith to become a role that can be assigned to others.

```
CREATE ROLE FOR USER JaneSmith;
```

The following statement creates the role Finance with MaryJones and JeffTurkott as role administrators with administrative rights (only) for the role.

```
CREATE ROLE Finance  
WITH ADMIN ONLY MaryJones, JeffTurkott;
```

The following example replaces the existing Finance role created in the previous example, replacing MaryJones and JeffTurkott with EllenChong and DaveLexx as role administrators, this time with exercise rights to the role.

```
CREATE OR REPLACE ROLE Finance  
WITH ADMIN EllenChong, DaveLexx;
```

Related Information

[Role administrators](#)

Role administrators are responsible for granting and revoking user-defined roles to users and other roles.

[User-extended roles](#)

A **user-extended role** is a user that has been extended to be a role, and is a type of user-defined role.

[Granting a role \(SQL Central\)](#)

Grant a role to a user or another role.

[min_role_admins option](#)

Sets the minimum number of administrators required to administer roles.

[DROP ROLE statement](#)

Removes a role from the database, or converts a user-extended role back to a regular user.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CREATE SCHEMA statement

Creates a collection of tables and views for a database user.

Syntax

```
CREATE SCHEMA  
AUTHORIZATION userid  
[ create-table-statement  
  | create-view-statement  
  | grant-statement  
] ... ;
```

Remarks

The CREATE SCHEMA statement creates a schema. A schema is a collection of tables and views along with their associated privileges.

The *userid* must be the user ID of the current connection. You cannot create a schema for another user.

If any statement contained in the CREATE SCHEMA statement fails, the entire CREATE SCHEMA statement is rolled back.

The CREATE SCHEMA statement is 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 privileges, you must use a REVOKE statement for each privilege granted.

The individual CREATE or GRANT statements are not separated by statement delimiters. The statement delimiter marks the end of the CREATE SCHEMA statement itself.

The individual CREATE or GRANT statements must be ordered such that the objects are created before privileges are granted on them.

Although you can create more than one schema for a user, doing so is not recommended.

Privileges

The system privileges required depend on the operation specified in the CREATE SCHEMA statement you define. For information about the required system privileges, see the System privileges sections for applicable statements (CREATE TABLE, CREATE VIEW, and GRANT).

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Core Feature. The ability to create multiple schemas for a single user is SQL optional Language Feature F171. The software does not support the use of REVOKE statements within the CREATE SCHEMA statement, and does not allow their use within Transact-SQL batches or procedures.
- **Transact-SQL** Supported by Adaptive Server Enterprise, which supports GRANT and REVOKE statements within the CREATE SCHEMA statement.

Example

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 the CREATE TABLE system privilege. 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 server. 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 );
```

Related Information

[CREATE TABLE statement](#)

Creates a new table in the database and, optionally, creates a table on a remote server.

[CREATE VIEW statement](#)

Creates a view on the database.

[GRANT statement](#)

Grant system and object-level privileges to users and roles.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CREATE SEMAPHORE statement

Creates or replaces a semaphore and establishes the initial value for its counter. A semaphore is a locking mechanism that uses a counter to communicate and control the availability of a resource such as an external library or procedure.

Syntax

```
CREATE [ OR REPLACE | TEMPORARY ] SEMAPHORE [ IF NOT EXISTS ] [  
  owner.]semaphore-name  
[ START WITH initial-count ]
```

Parameters

- **owner** The owner of the semaphore. *owner* can also be specified using an indirect identifier (for example, ``[@variable-name]``).
- **semaphore-name** The name of the semaphore. Specify a valid identifier in the CHAR database collation. *semaphore-name* can also be specified using an indirect identifier (for example, ``[@variable-name]``).
- **OR REPLACE clause** Use this clause to overwrite (update) the definition of a permanent semaphore of the same name, if one exists.

If the OR REPLACE clause is specified, and a semaphore with this name is in use at the time, then the statement returns an error.

Do not use this clause with the TEMPORARY or IF NOT EXISTS clauses.

- **TEMPORARY clause** Use this clause to create a temporary semaphore.
Do not use this clause with the OR REPLACE clause.
- **IF NOT EXISTS clause** Use this clause to create a semaphore only if it doesn't already exist. If a semaphore exists with the same name and same lifespan (permanent or temporary), then nothing happens and no error is returned.
Do not use this clause with the OR REPLACE clause.
- **START WITH clause** Use this clause to specify the initial value for the semaphore counter. If this clause is not specified, then *initial-count* is set to 0.

initial-count can be specified using a variable (for example, **START WITH @initial-count**).

If you set *initial-count* to NULL, or if it is set to a variable and the variable value is NULL, the behavior is equivalent to not specifying the clause.

Remarks

The CREATE SEMAPHORE statement creates a semaphore and establishes a counter for it. Each time a NOTIFY SEMAPHORE statement is executed, the counter for the associated semaphore is incremented. Each time a WAITFOR SEMAPHORE statement is executed, and assuming the current count is a positive integer, the counter for the associated semaphore is decremented.

Permanent and temporary mutexes and semaphores share the same namespace, therefore you cannot create two of these objects with the same name. Use of the OR REPLACE and IF NOT EXISTS clause can inadvertently cause an error related to naming. For example, if you have a permanent mutex, and you try to create a temporary semaphore with the same name, an error is returned

even if you specify IF NOT EXISTS. Similarly, if you have a temporary semaphore, and you try to replace it with a permanent semaphore with the same name by specifying OR REPLACE, an error is returned because this is equivalent to attempting to create a second object with the same name.

Permanent semaphore definitions persist across database restarts. However, their count returns to *initial-count* after a restart.

A temporary semaphore persists until the connection that created it is terminated, or until an explicit DROP operation is performed. If another connection is waiting for a temporary semaphore and the connection that created the temporary semaphore is terminated, then an error is returned to the waiting connection.

When replacing (OR REPLACE clause) a permanent semaphore, the old semaphore is deleted, and all connections waiting for the semaphore are notified.

If the OR REPLACE clause is specified, and a permanent semaphore with that name exists and connections are blocked waiting for the semaphore, the semaphore is still replaced. In this case, the waiting connections are unblocked and an error is returned to them indicating that the semaphore has been dropped. There is one exception however. If the replacement semaphore definition has identical settings, there is no impact to waiting connections.

A best practice is to avoid the use of the IF NOT EXISTS clause with the OR REPLACE clause.

Privileges

You must have the CREATE ANY MUTEX SEMAPHORE or CREATE ANY OBJECT system privilege.

Side effects

Automatic commit, but only for permanent semaphores.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statement creates a semaphore called license_counter and sets its counter to 3:

```
CREATE SEMAPHORE license_counter START WITH 3;
```

Related Information

[Mutexes and semaphores](#)

Use mutexes and semaphores in your application logic to achieve locking behavior and control and communicate the availability of resources.

[DROP SEMAPHORE statement](#)

Drops a semaphore.

[NOTIFY SEMAPHORE statement](#)

Increments the counter associated with a semaphore.

[WAITFOR SEMAPHORE statement](#)

Decrements the counter associated with a semaphore.

[SYSMUTEXSEMAPHORE system view](#)

Each row in the SYSMUTEXSEMAPHORE system view provides information about a user-defined mutex or semaphore in the database. The underlying system table for this view is ISYSMUTEXSEMAPHORE.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

CREATE SEQUENCE statement

Creates a sequence that can be used to generate primary key values that are unique across multiple tables, and for generating default values for a table.

Syntax

```
CREATE [ OR REPLACE ] SEQUENCE [ owner. ] sequence-name  
[ INCREMENT BY signed-integer ]  
[ START WITH signed-integer ]  
[ MINVALUE signed-integer | NO MINVALUE ]  
[ MAXVALUE signed-integer | NO MAXVALUE ]  
[ CACHE integer | NO CACHE ]  
[ CYCLE | NO CYCLE ]
```

Parameters

OR REPLACE clause Specifying OR REPLACE creates a new sequence, or replaces an existing sequence with the same name. If you do not use the OR REPLACE clause, an error is returned if you specify the name of a sequence that already exists for the current user.

INCREMENT BY clause Defines the amount the next sequence value is incremented from the last value assigned. The default is 1. Specify a negative value to generate a descending sequence. An error is returned if the INCREMENT BY value is 0.

START WITH clause Defines the starting sequence value. If you do not specify a value for the START WITH clause, MINVALUE is used for ascending sequences and MAXVALUE is used for descending sequences. An error is returned if the START WITH value is beyond the range specified by MINVALUE or MAXVALUE.

MINVALUE clause Defines the smallest value generated by the sequence. The default is 1. An error is returned if MINVALUE is greater than $(2^{63}-1)$ or less than $-(2^{63}-1)$. An error is also returned if MINVALUE is greater than MAXVALUE.

MAXVALUE clause Defines the largest value generated by the sequence. The default is $2^{63}-1$. An error is returned if MAXVALUE is greater than $2^{63}-1$ or less than $-(2^{63}-1)$.

CACHE clause Specifies the number of preallocated sequence values that are kept in memory for faster access. When the cache is exhausted, the sequence cache is repopulated and a corresponding entry is written to the transaction log. At checkpoint time, the current value of the cache is forwarded to the ISYSEQUENCE system table. The default is 100.

CYCLE clause Specifies whether values should continue to be generated after the maximum or minimum value is reached.

The default is NO CYCLE, which returns an error once the maximum or minimum value is reached.

Remarks

A sequence is a database object that allows the automatic generation of numeric values. A sequence is not bound to a specific or unique table column and is only accessible through the table column to which it is applied.

Sequences can generate values in one of the following ways:

- Increment or decrement monotonically without bound
- Increment or decrement monotonically to a user-defined limit and stop
- Increment or decrement monotonically to a user-defined limit and cycle back to the beginning and start again

You control the behavior when the sequence runs out of values using the CYCLE clause.

If a sequence is increasing and it exceeds the MAXVALUE, MINVALUE is used as the next sequence value if CYCLE is specified. If a sequence is decreasing and it falls below MINVALUE, MAXVALUE is used as the next sequence value if CYCLE is specified. If CYCLE is not specified, an error is returned.

Sequence values cannot be used with views or materialized view definitions.

Privileges

You must have the CREATE ANY SEQUENCE or CREATE ANY OBJECT system privilege to create sequences.

To replace an existing sequence, you must have one of the following:

- CREATE ANY SEQUENCE and DROP ANY SEQUENCE system privileges.
- CREATE ANY OBJECT and DROP ANY OBJECT system privileges.
- ALTER ANY OBJECT or ALTER ANY SEQUENCE system privileges.

Side effects

None

Standards

- **ANSI/ISO SQL Standard** Sequences comprise SQL Language Feature T176. The software does not allow optional specification of the sequence data type. This behavior can be achieved with a CAST when using the sequence.

In addition, the following are not in the standard:

- CACHE clause
- OR REPLACE syntax
- CURRVAL expression
- Use of sequences in DEFAULT expressions

Example

The following example creates a sequence named Test that starts at 4, increments by 2, does not cycle, and caches 15 values at a time:

```
CREATE SEQUENCE Test
START WITH 4
INCREMENT BY 2
NO MAXVALUE
NO CYCLE
CACHE 15;
```

Related Information

[Use of a sequence to generate unique values](#)

You can use a **sequence** to generate values that are unique across multiple tables or that are different from a set of natural numbers.

[ALTER SEQUENCE statement](#)

Alters a sequence.

[DROP SEQUENCE statement](#)

Drops a sequence.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)» [Alphabetical list of SQL statements](#)

CREATE SERVER statement

Creates a remote server or a directory access server.

Syntax

- Create a remote server

```

CREATE [ REMOTE ] SERVER server-name
CLASS server-class-string | variable
USING connection-info-string | variable
[ READ ONLY [ ON | OFF | VALUE variable ] ]
[ DEFAULT LOGIN string | variable [ IDENTIFIED BY string | variable ] ]

```

server-class-string :

```

{ 'ADSODBC' | 'ADS_ODBC'
| 'ASEODBC' | 'ASE_ODBC'
| 'DB2ODBC' | 'DB2_ODBC'
| 'HANAODBC' | 'HANA_ODBC'
| 'IQODBC' | 'IQ_ODBC'
| 'MIRROR'
| 'MSACCESSODBC' | 'MSACCESS_ODBC'
| 'MSSODBC' | 'MSS_ODBC'
| 'MYSQLODBC' | 'MYSQL_ODBC'
| 'ODBC'
| 'ORAODBC' | 'ORA_ODBC'
| 'SAODBC' | 'SA_ODBC'
| 'ULODBC' | 'UL_ODBC' }

```

connection-info-string :

```

{ 'data-source-name' | 'sqlanywhere-connection-string' }

```

- Create a directory access server

```

CREATE SERVER server-name
CLASS 'DIRECTORY'
USING using-string | variable
[ READ ONLY [ ON | OFF | VALUE variable ] ]
[ ALLOW { 'ALL' | 'SPECIFIC' | variable } USERS ]

```

using-string :

```

'ROOT = path [ ;SUBDIRS = n ] [ ;CREATEDIRS = { YES | NO } ] [ ;DELIMITER =
{ / | \ } ]'

```

- Create a remote server (SAP HANA syntax)

```

CREATE REMOTE SOURCE remote-source-name
ADAPTER adapter-name | variable
CONFIGURATION connection-info-string | variable
[ READ ONLY [ ON | OFF | VALUE variable ] ]
[ WITH CREDENTIAL TYPE { 'PASSWORD' | variable } USING { 'USER=remote-
user;password=remote-password' | variable }

```

Parameters

CREATE [REMOTE] SERVER The REMOTE keyword is for remote servers, is optional, and is provided for compatibility with other databases.

CLASS clause Specifies the server class you want to use for a remote connection. Server classes contain detailed server capability information.

The DIRECTORY class is used to create a directory access server that accesses a directory on the local computer.

The server classes are:

- **SAODBC** for SQL Anywhere.
- **ULODBC** for UltraLite.

Note You cannot create a remote server for an UltraLite database running on Mac OS X.

- **ADSODBC** for Advantage Database Server.
- **ASEODBC** for SAP Adaptive Server Enterprise (version 10 and later).
- **DB2ODBC** for IBM DB2.
- **HANAODBC** for SAP HANA.
- **IQODBC** for SAP IQ.
- **MSACCESSODBC** for Microsoft Access.
- **MSSODBC** for Microsoft SQL Server.
- **MYSQLODBC** for Oracle MySQL.
- **ODBC** for all other ODBC data sources.
- **ORAODBC** for Oracle Database servers (version 8.0 and later).

Note When using remote data access, if you use an ODBC driver that does not support Unicode, then character set conversion is not performed on data coming from that ODBC driver.

READ ONLY clause (remote server) Specifies that the remote server is accessed in read-only mode. If the clause is not specified, or if READ ONLY OFF is specified, then the remote server is not accessed in read-only mode. If READ ONLY or READ ONLY ON is specified, then the remote server is accessed in read-only mode.

READ ONLY clause (directory access server) Specifies whether the files accessed by the directory are read-only and cannot be modified. By default, READ ONLY is set to NO.

ALLOW USERS clause (directory access servers) Specify ALLOW 'SPECIFIC' USERS to restrict access to the directory access server to users with an external login. You must explicitly create an external login to the directory access server for each user that needs access. This clause is the default.

Specify the ALLOW 'ALL' USERS clause if you are not concerned about who has access to the directory access server, or you want everyone in your database to have access. This clause creates a default external login for the directory access server that is available to all users.

USING clause (remote servers) When you create a remote server, the USING clause supplies a connection string for the database server. The appropriate connection string depends on the driver being used, which in turn depends on the value specified.

The USING clause is an ODBC connection string that can include '*DSN=data-source-name*' to specify an ODBC data source name and/or '*DRIVER=driver-name*' to specify a driver binary on Unix or a driver name on Windows.

For SQL Anywhere remote servers (SAODBC server classes), the *connection-info-string* parameter can be any valid connection string. Use any supported connection parameter. For example, if you have connection problems, then include a LOG connection parameter to troubleshoot the connection attempt.

The string in the USING clause can also contain local or global variable names enclosed in braces ({*variable-name*}). The SQL variable name must be of type CHAR, VARCHAR, or LONG VARCHAR. For example, a USING clause that contains '*DSN={@mydsn}*' indicates that *@mydsn* is a SQL variable and that the current contents of the *@mydsn* variable should be substituted when a connection is made to the remote data access server.

USING clause (directory access servers) Specify the server connection information.

When you create a directory access server, the USING clause specifies the following values for the local directory:

- **ROOT clause** Specifies the path, relative to the database server, that is the root of the directory access class. When you create a proxy table using the directory access server name, the proxy table is relative to this root path.
- **SUBDIRS clause** Specifies a number between 0 and 10 that represents the number of levels of directories within the root that the database server can access. If SUBDIRS is omitted or set to 0, then only the files in the root directory are accessible via the directory access server. You can create proxy tables to any of the directories or subdirectories available via the directory access server.
- **CREATEDIRS clause** Specifies whether directories can be created using the directory access server. The default is NO.
- **DELIMITER clause** Specifies whether paths are delimited by a slash (/) or backslash (\) character. By default, the native path delimiter is used.

The string in the USING clause can also contain local or global variable names enclosed in braces ({*variable-name*}). The SQL variable name must be of type CHAR, VARCHAR, or LONG VARCHAR. For example, a USING clause that contains '*R00T={@mypath}*' indicates that *@mypath* is a SQL variable and that the current contents of the *@mypath* variable should be substituted when a connection to the directory access server is established.

DEFAULT LOGIN clause (remote server) Specifies a default user ID and (optionally) password for an account on the remote server that is to be used the default login. Values for the DEFAULT LOGIN clause are restricted to 128 bytes.

- **IDENTIFIED BY clause** The IDENTIFIED BY clause specifies the remote password for the remote user. This value can be either a string or a variable. The remote user and remote password combination must be valid on the remote-server.

If you omit the IDENTIFIED BY clause, the password is sent to the remote server as NULL. However, if you specify IDENTIFIED BY '' (an empty string), then the password sent is the empty string.

If you use this statement in a procedure, do not specify the password (IDENTIFIED BY clause) as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

CREATE REMOTE SOURCE (SAP HANA syntax) This syntax, including the parameters and their values, is semantically equivalent to the syntax for creating a remote server (CREATE [REMOTE]) syntax and is provided for compatibility with SAP HANA servers. There is a one-to-one clause match between the two syntaxes as follows:

- **ADAPTER *adapter-name*** See the CLASS clause description for the CREATE [REMOTE] SERVER syntax.
- **CONFIGURATION *connection-info-string*** See the USING clause description for the CREATE [REMOTE] SERVER syntax.
- **READ ONLY [ON | OFF] clause** See the READ ONLY clause description for the CREATE [REMOTE] SERVER syntax (remote servers).
- **WITH CREDENTIAL TYPE clause** See the DEFAULT LOGIN clause description for the CREATE [REMOTE] SERVER syntax.

Remarks (remote server)

Use the CREATE SERVER statement to create a remote server to access to data in other data sources. Once you create a remote server, you must create local proxy tables to map to the remote tables. Database users use proxy tables to access the contents of the remote tables. Create an external login for each database user that needs to communicate with the remote server.

Connections to a remote server are first attempted using the current executing user's external login. If this user does not have an external login, then the connection is attempted using the DEFAULT LOGIN credentials. If the remote server was created without a DEFAULT LOGIN, and no external login has been defined for the user, then the connection is attempted with the current executing user ID and password.

When running procedures that involve connections to a remote server, you can use the `extern_login_credentials` option to specify whether the remote data access connections are performed using the external login credentials of the logged in user or the effective user.

On Unix, the database server runs as a specific user, so file permissions are based on the privileges granted to the database server user.

When accessing data from a remote server, if you use an ODBC driver that does not support Unicode, then character set conversion is not performed on data coming from that ODBC drive.

When you define a remote server, an entry is added to the ISYSSERVER system table for the remote server. View the list of remote servers by querying the SYSSERVER system view.

Note For *required* parameters that accept variable names, the database server returns an error if any of the following conditions is true:

- The variable does not exist
- The contents of the variable are NULL
- The variable exceeds the length allowed by the parameter
- The data type of the variable does not match that required by the parameter

Remarks (directory access server)

Use the CREATE SERVER statement to create a directory access server that accesses the local directory structure on the computer where the database server is running. Create an external login for each database user that needs to use the directory access server. Once you create a directory access server, you must create a proxy table for it. Database users use proxy tables to access the

contents of a directory on the database server's local file system.

View the list of directory access servers by querying the SYSSERVER system view.

Privileges

You must have the SERVER OPERATOR system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example creates a SQL Anywhere remote server named RemoteSA using the SQL Anywhere ODBC driver:

```
CREATE SERVER RemoteSA
CLASS 'SAODBC'
USING 'DRIVER=SQL Anywhere 17;DSN=RemoteDS';
```

The following example directly loads the SQL Anywhere ODBC driver without using the ODBC driver manager:

```
CREATE SERVER RemoteSA
CLASS 'SAODBC'
USING 'DRIVER=SQL Anywhere Native;DSN=RemoteDS';
```

The following example uses a variable reference to create a dynamic remote data access server. You need the MANAGE ANY USER and CREATE TABLE system privileges to run this example.

```
CREATE SERVER RemoteSA
CLASS 'SAODBC'
USING 'DRIVER=SQL Anywhere 17;DSN={dsn_string};Server=saremot;UID=DBA;PWD=sql';

CREATE VARIABLE dsn_string LONG VARCHAR;
SET dsn_string = 'Test17';

CREATE EXTERNLOGIN DBA TO RemoteSA;

CREATE EXISTING TABLE test_employees
AT 'RemoteSA...Employees';
SELECT * FROM test_employees;
DROP REMOTE CONNECTION TO RemoteSA CLOSE ALL;
```

The following example creates an Adaptive Server Enterprise (ASE) remote server named ase_prod using the ASE ODBC driver:

```
CREATE SERVER ase_prod
CLASS 'ASEODBC'
USING 'DSN=remoteASE';
```

The following example creates a remote server for the Oracle server named oracle723. Its ODBC data source name is oracle723.

```
CREATE SERVER oracle723
CLASS 'ORAODBC'
USING 'oracle723';
```

The following example creates a directory access server that only sees files within the directory c:\temp:

```
CREATE SERVER diskserver0
CLASS 'DIRECTORY'
USING 'ROOT=c:\\temp';
CREATE EXTERNLOGIN DBA TO diskserver0;
CREATE EXISTING TABLE diskdir0 AT 'diskserver0;;;.';

-- Get a list of those files.
SELECT privileges, file_name, size FROM diskdir0;
```

The following example creates a dynamic directory access server that is used to explore two different directories:

```
CREATE SERVER diskserver9
CLASS 'DIRECTORY'
USING '{dir_options}';

CREATE EXTERNLOGIN DBA TO diskserver9;
CREATE EXISTING TABLE diskdir9 AT 'diskserver9;;;.';

CREATE VARIABLE dir_options VARCHAR(256);
SET dir_options = 'ROOT=c:\\temp;SUBDIRS=9;DELIMITER=/';
SELECT * FROM diskdir9;

DROP REMOTE CONNECTION TO diskserver9 CLOSE ALL;
SET dir_options = 'ROOT=c:\\ProgramData;SUBDIRS=9;DELIMITER=/';
SELECT * FROM diskdir9;
```

When you create a remote server, to bypass the ODBC driver manager, use the syntax below, followed by the remainder of the *connection-info-string*:

```
CREATE SERVER remote-server
CLASS 'SAODBC'
USING 'DRIVER=SQL Anywhere Native;DSN=my-dsn;UID=my-username;PWD=my-pwd';
```

This syntax allows remote data access to load the SQL Anywhere ODBC driver directly and is

supported by Windows and Unix. Loading the SQL Anywhere ODBC driver directly ensures that the ODBC driver for the current server version is used. Also, if the SQL Anywhere ODBC driver is only used for remote data access, then it does not need to be registered.

On Unix platforms you can also reference the SQL Anywhere ODBC driver. The syntax is as follows:

```
USING 'DRIVER=SQL Anywhere 17;DSN=my-dsn'
```

Note If the application also makes use of non-SQL Anywhere remote servers, or if there are SQL Anywhere remote servers defined without using 'DRIVER=SQL Anywhere Native', then remote data access still uses a driver manager for the other remote servers.

Related Information

[Remote data access](#)

Remote data access gives you access to data in other data sources as well as access to the files on the computer that is running the database server.

[Remote servers and remote table mappings](#)

SQL Anywhere presents tables to a client application as if all the data in the tables were stored in the database to which the application is connected. Before you can map remote objects to a local proxy table, you must define the remote server where the remote object is located.

[Directory access servers](#)

A **directory access server** is a remote server that gives you access to the local file structure of the computer running the database server.

[Server classes for remote data access](#)

The server class you specify in the CREATE SERVER statement determines the behavior of a remote connection.

[Connection parameters](#)

Connection parameters are included in connection strings.

[Creating directory access servers \(SQL\)](#)

Create directory access servers using the CREATE SERVER statement.

[extern_login_credentials option](#)

Controls whether the external login credentials of the session user or the executing (effective) user are used when making remote connections. This option is provided for backwards compatibility. For security reasons, do not specify this option.

[ALTER SERVER statement](#)

Modifies the attributes of a remote server.

[CREATE EXTERNLOGIN statement](#)

Assigns an alternate login name and password to be used when communicating with a remote server.

[CREATE EXISTING TABLE statement](#)

Creates a new proxy table, which represents an existing object on a remote server.

[DROP SERVER statement](#)

Drops a remote server from the catalog.

[DROP REMOTE CONNECTION statement](#)

Drops remote data access connections to a remote server.

[SYSSERVER system view](#)

Each row in the SYSSERVER system view describes a remote server. The underlying system table for this view is ISYSSERVER.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

CREATE SERVICE statement [HTTP web service]

Creates a new HTTP web service.

Syntax

```
CREATE [ OR REPLACE ] SERVICE service-name
TYPE { 'RAW' | 'HTML' | 'JSON' | 'XML' }
URL [PATH] { ON | OFF | ELEMENTS } ]
[ common-attributes ]
[ AS { statement | NULL } ]
```

common-attributes :

```
[ AUTHORIZATION { ON | OFF } ]
[ ENABLE | DISABLE ]
[ METHODS 'method,...' ]
[ SECURE { ON | OFF } ]
[ USER { user-name | NULL } ]
```

method :

```
DEFAULT
| POST
| GET
| HEAD
| OPTIONS
| PUT
| DELETE
| NONE
| *
```

Parameters

service-name Web service names can be any sequence of alphanumeric characters or slash (/), hyphen (-), underscore (_), period (.), exclamation mark (!), tilde (~), asterisk (*), apostrophe ('), left parenthesis ((), or right parenthesis ()), except that the service name must not begin or end with a slash (/) or contain two or more consecutive slashes (for example, //).

You can name your service root, but this name has a special function.

TYPE clause Identifies the type of the service where each service defines a specific response format. The type must be one of the listed service types. There is no default value.

- **'RAW'** The result set is sent to the client without any formatting. Utilization of this service requires that all content markup is explicitly provided. Complex dynamic content containing current content with markup, JavaScript and images can be generated on demand. The media type may be specified by setting the Content-Type response header using the `sa_set_http_header` procedure. Setting the Content-Type header to 'text/html' is good practice when generating HTML markup to ensure that all browsers display the markup as HTML and not text/plain.
- **'HTML'** The result set is returned as an HTML representation of a table or view.
- **'JSON'** The result set is returned in JavaScript Object Notation (JSON). A JSON service

does not automatically process JSON input. It only presents data (in the response) in JSON format. JSON accepts POST/PUT methods where application/x-www-form-urlencoded is supported. If for a POST/PUT method, **Content-Type: application/json** is specified, then the application may use `http_variable('body')` to retrieve the JSON (request) content. The database server does not parse the JSON input automatically. It is up to the application to parse it.

- **'XML'** The result set is returned as XML. If the result set is already XML, no additional formatting is applied. Otherwise, it is automatically formatted as XML. As an alternative approach, a RAW service could return a select using the FOR XML RAW clause having set a valid Content-Type such as text/xml using `sa_set_http_header` procedure.

URL clause Determines whether URL paths are accepted and, if so, how they are processed. Specifying URL PATH has the same effect as URL.

- **OFF** Indicates that the service name in a URL request must not be followed by a path. OFF is the default setting. For example, the following form is disallowed due to the path elements `/aaa/bbb/cc`.

`http://host-name/service-name/aaa/bbb/cc`

Suppose that `CREATE SERVICE echo URL PATH OFF` was specified when creating the web service. A URL similar to `http://localhost/echo?id=1` produces the following values:

Function call	Result
<code>HTTP_VARIABLE('id')</code>	1
<code>HTTP_HEADER('@HttpQueryString')</code>	id=1

- **ON** Indicates that the service name in a URL request can be followed by a path. The path value is returned by querying a dedicated HTTP variable named URL. A service can be defined to explicitly provide the URL parameter or it may be retrieved using the `HTTP_VARIABLE` function. For example, the following form is allowed:

`http://host-name/service-name/aaa/bbb/cc`

Suppose that `CREATE SERVICE echo URL PATH ON` was specified when creating the web service. A URL similar to `http://localhost/echo/one/two?id=1` produces the following values:

Function call	Result
<code>HTTP_VARIABLE('id')</code>	1
<code>HTTP_VARIABLE('URL')</code>	one/two
<code>HTTP_HEADER('@HttpQueryString')</code>	id=1

- **ELEMENTS** Indicates that the service name in a URL request may be followed by a path. The path is obtained in segments by specifying a single parameter keyword **URL1**, **URL2**, and so on. Each parameter may be retrieved using the `HTTP_VARIABLE` or `NEXT_HTTP_VARIABLE` functions. These iterator functions can be used in applications where a variable number of path elements can be provided. For example, the following form is allowed:

`http://host-name/service-name/aaa/bbb/cc`

Suppose that `CREATE SERVICE echo URL PATH ELEMENTS` was specified when creating the web

service. A URL similar to <http://localhost/echo/one/two?id=1> produces the following values:

Function call	Result
HTTP_VARIABLE('id')	1
HTTP_VARIABLE('URL1')	one
HTTP_VARIABLE('URL2')	two
HTTP_VARIABLE('URL3')	NULL
HTTP_HEADER('@HttpQueryString')	id=1

Up to 10 elements can be obtained. A NULL value is returned if the corresponding element is not supplied. In the above example, `HTTP_VARIABLE('URL3')` returns NULL because no corresponding element was supplied.

AUTHORIZATION clause Determines whether users must specify a user name and password through basic HTTP authorization when connecting to the service. The default value is ON. If authorization is OFF, the AS clause is required for all services and a user must be specified with the USER clause. All requests are run using that user's account and privileges. If AUTHORIZATION is ON, all users must provide a user name and password. Optionally, you can limit the users that are permitted to use the service by providing a user or group name with the USER clause. If the user name is NULL, all known users can access the service. The AUTHORIZATION clause allows your web services to use database authorization and privileges to control access to the data in your database.

When the authorization value is ON, an HTTP client connecting to a web service uses basic authentication (RFC 2617) that obfuscates the user and password information using base-64 encoding. Use the HTTPS protocol for increased security.

ENABLE and DISABLE clauses Determines whether the service is available for use. By default, when a service is created, it is enabled. When creating or altering a service, you may include an ENABLE or DISABLE clause. Disabling a service effectively takes the service off line. Later, it can be enabled using ALTER SERVICE with the ENABLE clause. An HTTP request made to a disabled service typically returns a [404 Not Found](#) HTTP status.

METHODS clause Specifies the HTTP methods that are supported by the service. Valid values are DEFAULT, POST, GET, HEAD, OPTIONS, PUT, DELETE, and NONE. An asterisk (*) may be used as a short form to represent the POST, GET, and HEAD methods which are default request types for the RAW, HTML, and XML service types. Not all HTTP methods are valid for all the service types. The following table summarizes the valid HTTP methods that can be applied to each service type:

Method value	Applies to service	Description
DEFAULT	all	Use DEFAULT to reset the set of default HTTP methods for the given service type. It cannot be included in a list with other method values.
POST	RAW, HTML, JSON, XML	Enabled by default.
GET	RAW, HTML, JSON, XML	Enabled by default.
HEAD	RAW, HTML, JSON, XML	Enabled by default.
OPTIONS	RAW, HTML, JSON, XML	Not enabled by default.
PUT	RAW, HTML, JSON, XML	Not enabled by default.

Method value	Applies to service	Description
DELETE	RAW, HTML, JSON, XML	Not enabled by default.
NONE	all	Use NONE to disable access to a service.
*	RAW, HTML, JSON, XML	Same as specifying 'POST,GET,HEAD' .

For example, you can use either of the following clauses to specify that a service supports all HTTP method types:

```
METHODS '*' ,OPTIONS,PUT,DELETE'
METHODS 'POST,GET,HEAD,OPTIONS,PUT,DELETE'
```

To reset the list of request types for any service type to its default, you can use the following clause:

```
METHODS 'DEFAULT'
```

SECURE clause Specifies whether the service should be accessible on a secure or non-secure listener. ON indicates that only HTTPS connections are accepted, and that connections received on the HTTP port are automatically redirected to the HTTPS port. OFF indicates that both HTTP and HTTPS connections are accepted, provided that the necessary ports are specified when starting the web server. The default value is OFF.

USER clause Specifies a database user, or group of users, with privileges to execute the web service request. A USER clause must be specified when the service is configured with AUTHORIZATION OFF and should be specified with AUTHORIZATION ON (the default). An HTTP request made to a service requiring authorization results in a [401 Authorization Required](#) HTTP response status. Based on this response, the web browser prompts for a user ID and password.

Caution

It is strongly recommended that you specify a USER clause when authorization is enabled (default). Otherwise, authorization is granted to all users.

The USER clause controls which database user accounts can be used to process service requests. Database access permissions are restricted to the privileges assigned to the user of the service.

statement Specifies a command, such as a stored procedure call, to invoke when the service is accessed.

An HTTP request to a non-DISH service with no *statement* specifies the SQL expression to execute within its URL. Although authorization is required, this capability should not be used in production systems because it makes the server vulnerable to SQL injections. When a statement is defined within the service, the specified SQL statement is the only statement that can be executed through the service.

In a typical web service application, you use *statement* to call a function or procedure. You can pass host variables as parameters to access client-supplied HTTP variables.

The following *statement* demonstrates a procedure call that passes two host variables to a procedure named AuthenticateUser. This call presumes that a web client supplies the user_name

and user_password variables:

```
CALL AuthenticateUser ( :user_name, :user_password );
```

Remarks

Service definitions are stored within the ISYSWEBSERVICE table and can be examined from the SYSWEBSERVICE system view.

If a USER clause is specified, the service is dropped if *user-name* is dropped.

Privileges

You must have the MANAGE ANY WEB SERVICE system privilege.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.
- **Transact-SQL** CREATE SERVICE is supported by Adaptive Server Enterprise, for XML and RAW types only.

Example

The following example demonstrates how to create a JSON service.

Start a database server with the -xs (http or https) option and then execute the following SQL statements to set up the service:

```
CREATE PROCEDURE ListEmployees()  
RESULT (  
  EmployeeID      integer,  
  Surname          person_name_t,  
  GivenName        person_name_t,  
  StartDate        date,  
  TerminationDate  date )  
BEGIN  
  SELECT EmployeeID, Surname, GivenName, StartDate, TerminationDate  
  FROM GROUP0.Employees  
END;  
  
CREATE SERVICE "jsonEmployeeList"  
  TYPE 'JSON'  
  AUTHORIZATION OFF  
  SECURE OFF  
  USER DBA  
  AS CALL ListEmployees();
```

The JSON service provides data for easy consumption by an AJAX call back.

Execute the following SQL statement to create an HTML service that provides the service in a readable form:

```
CREATE SERVICE "EmployeeList"  
  TYPE 'HTML'  
  AUTHORIZATION OFF  
  SECURE OFF  
  USER DBA  
  AS CALL ListEmployees();
```

Use a web browser to access the service using a URL similar to <http://localhost/EmployeeList>.

Related Information

[How to access client-supplied HTTP variables and headers](#)

Variables and headers in an HTTP client request can be accessed by web services.

[The database server as an HTTP web server](#)

The database server contains a built-in HTTP web server that allows you to provide online web services from a database.

[How to create and customize a root web service](#)

An HTTP client request that does not match any web service request is processed by the **root** web service if a **root** web service is defined.

[How to develop web service applications in an HTTP web server](#)

Customized web pages can be produced by the stored procedures that are called from web services. A very elementary web page example is shown below.

[How to browse a SQL Anywhere HTTP web server](#)

How your web services are named and designed dictates what constitutes a valid URL.

[ALTER SERVICE statement \[HTTP web service\]](#)

Alters an existing HTTP web service.

[DROP SERVICE statement](#)

Drops a web service.

[sp_parse_json system procedure](#)

Returns a representation of JSON data using SQL data types.

[SYSWEBSERVICE system view](#)

Each row in the SYSWEBSERVICE system view holds a description of a web service. The underlying system table for this view is ISYSWEBSERVICE.

[sa_set_http_header system procedure](#)

Permits a web service to set an HTTP response header.

[Identifiers](#)

Identifiers are the names of objects in the database, such as user IDs, tables, and columns.

[ROW constructor \[Composite\]](#)

Returns a sequence of (*field name data type*, ...) pairs named **fields**.

[ARRAY constructor \[Composite\]](#)

Returns elements of a specific data type.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)» [Alphabetical list of SQL statements](#)

CREATE SERVICE statement [SOAP web service]

Creates a new SOAP over HTTP or DISH service.

Syntax

- **SOAP over HTTP services**

```

CREATE [ OR REPLACE ] SERVICE service-name
TYPE 'SOAP'
[ DATATYPE { ON | OFF | IN | OUT } ]
[ FORMAT { 'DNET' | 'CONCRETE' [ EXPLICIT { ON | OFF } ] | 'XML' | NULL } ]
[ common-attributes ]
AS statement

```

common-attributes :

```

[ AUTHORIZATION { ON | OFF } ]
[ ENABLE | DISABLE ]
[ METHODS 'method,...' ]
[ SECURE { ON | OFF } ]
[ USER { user-name | NULL } ]

```

method :

```

DEFAULT
| POST
| HEAD
| NONE

```

- **DISH services**

```

CREATE SERVICE service-name
TYPE 'DISH'
[ GROUP { group-name | NULL } ]
[ FORMAT { 'DNET' | 'CONCRETE' [ EXPLICIT { ON | OFF } ] | 'XML' | NULL } ]
[ common-attributes ]

```

common-attributes:

```

[ AUTHORIZATION { ON | OFF } ]
[ ENABLE | DISABLE ]
[ METHODS 'method,...' ]
[ SECURE { ON | OFF } ]
[ USER { user-name | NULL } ]

```

method :

```

DEFAULT
| POST
| GET
| HEAD
| NONE
| *

```

Parameters

service-name Web service names can be any sequence of alphanumeric characters or slash (/), hyphen (-), underscore (_), period (.), exclamation mark (!), tilde (~), asterisk (*), apostrophe ('), left parenthesis ((), or right parenthesis ()), except that the service name must not begin or end with a slash (/) or contain two or more consecutive slashes (for example, //).

Unlike other services, you cannot use a slash (/) anywhere in a DISH service name.

You can name your service root, but this name has a special function.

TYPE clause Identifies the type of the service where each service defines a specific response format. The type must be one of the listed service types. There is no default value.

- o **'SOAP'** The result set is returned as an XML payload known as a SOAP envelope. The format of the data may be further refined using by the FORMAT clause. A request to a SOAP service must be a valid SOAP request, not just a general HTTP request.
- o **'DISH'** A DISH service (Determine SOAP Handler) is a SOAP endpoint that references any SOAP service within its GROUP context. It also exposes the interfaces to its SOAP services by generating a WSDL (Web Services Description Language) for consumption by SOAP client toolkits.

GROUP clause A DISH service without a GROUP clause exposes all SOAP services defined within the database. By convention, the SOAP service name can be composed of a GROUP and a NAME element. The name is delimited from the group by the last slash character. For example, a SOAP service name defined as 'aaa/bbb/ccc' is 'ccc', and the group is 'aaa/bbb'. Delimiting a DISH service using this convention is invalid. Instead, a GROUP clause is applied to specify the group of SOAP services for which it is to be the SOAP endpoint.

Note Slashes are converted to underscores within the WSDL to produce valid XML. Use caution when using a DISH service that does not specify a GROUP clause such that it exposes all SOAP services that may contain slashes. Use caution when using groups with SOAP service names that contain underscores to avoid ambiguity.

DATATYPE clause Applies to SOAP services only. When DATATYPE OFF is specified, SOAP input parameters and response data are defined as XMLSchema string types. In most cases, true data types are preferred because it negates the need for the SOAP client to cast the data prior to computation. Parameter data types are exposed in the schema section of the WSDL generated by the DISH service. Output data types are represented as XML schema type attributes for each column of data.

The following values are permitted for the DATATYPE clause:

- o **ON** Generates data typing of input parameters and result set responses.
- o **OFF** All input parameters and response data are typed as XMLSchema string (default).
- o **IN** Generates true data types for input parameters only. Response data types are XMLSchema string.
- o **OUT** Generates true data types for responses only. Input parameters are typed as XMLSchema string.

FORMAT clause This clause specifies the output format when sending responses to SOAP client applications.

The SOAP service format is dictated by the associated DISH service format specification when it is not specified by the SOAP service. The default format is DNET.

SOAP requests should be directed to the DISH service (the SQL Anywhere SOAP endpoint) to leverage common formatting rules for a group of SOAP services (SOAP operations). A SOAP service FORMAT specification overrides that of a DISH service. The format specification of the DISH service is used when a SOAP service does not define a FORMAT clause. If no FORMAT is provided by either service then the default is 'DNET'.

The following formats are supported:

- o **'DNET'** The output is in a System.Data.DataSet compatible format for consumption by .NET client applications. (default)
- o **'CONCRETE'** This output format is used to support client SOAP toolkits that are capable of generating interfaces representing arrays of row and column objects but are not able to consume the DNET format. Java and .NET clients can easily consume this output format.

The specific format is exposed within the WSDL as an explicit dataset object or a SimpleDataset. Both dataset representations describe a data structure representing an array of rows with each row containing an array of columns. An explicit dataset object has the advantage of representing the actual shape of the result set by providing parameter names and datatypes for each column in the row. In contrast, the SimpleDataset exposes rows containing an unbounded number of columns of any type.

FORMAT 'CONCRETE' EXPLICIT ON requires that the Service statement calls a stored procedure which defines a RESULT clause. Having met this condition, the SOAP service will expose an explicit dataset whose name begins with the service name appended with Dataset.

If the condition is not met, a SimpleDataset is used.

- o **'XML'** The output is generated in an XMLSchema string format. The response is an XML document that requires further processing by the SOAP client to extract column data. This format is suitable for SOAP clients that cannot generate intermediate interface objects that represent arrays of rows and columns.
- o **NULL** A NULL type causes the SOAP or DISH service to use the default behavior. The format type of an existing service is overwritten when using the NULL type in an ALTER SERVICE statement.

AUTHORIZATION clause Determines whether users must specify a user name and password through basic HTTP authorization when connecting to the service. The default value is ON. If authorization is OFF, the AS clause is required for SOAP services, and a user must be specified with the USER clause. All requests are run using that user's account and privileges. If AUTHORIZATION is ON, all users must provide a user name and password. Optionally, you can limit the users that are permitted to use the service by providing a user or group name with the USER clause. If the user name is NULL, all known users can access the service. The AUTHORIZATION clause allows your web services to use database authorization and privileges to control access to the data in your database.

When the authorization value is ON, an HTTP client connecting to a web service uses basic authentication (RFC 2617) which obfuscates the user and password information using base-64 encoding. It is recommended that you use the HTTPS protocol for increased security.

ENABLE and DISABLE clauses Determines whether the service is available for use. By default, when a service is created, it is enabled. When creating or altering a service, you may include an ENABLE or DISABLE clause. Disabling a service effectively takes the service off line. Later, it can be enabled using ALTER SERVICE with the ENABLE clause. An HTTP request made to

a disabled service typically returns a [404 Not Found](#) HTTP status.

METHODS clause Specifies the HTTP methods that are supported by the service. Valid values are DEFAULT, POST, GET, HEAD, and NONE. An asterisk (*) may be used as a short form to represent the POST, GET, and HEAD methods. The default method types for SOAP services are POST and HEAD. The default method types for DISH services are GET, POST, and HEAD. Not all HTTP methods are valid for all the service types. The following table summarizes the valid HTTP methods that can be applied to each service type:

Method value	Applies to service	Description
DEFAULT	both	Use DEFAULT to reset the set of default HTTP methods for the given service type. It cannot be included in a list with other method values.
POST	both	Enabled by default for SOAP.
GET	DISH only	Enabled by default for DISH.
HEAD	both	Enabled by default for SOAP and DISH.
NONE	both	Use NONE to disable access to a service. When applied to a SOAP service, the service cannot be directly accessed by a SOAP request. This enforces exclusive access to a SOAP operation through a DISH service SOAP endpoint. It is recommended that you specify METHODS 'NONE' for each SOAP service.
*	DISH only	Same as specifying 'POST,GET,HEAD' .

For example, you can use the following clause to specify that a service supports all SOAP over HTTP method types:

```
METHODS 'POST,HEAD'
```

To reset the list of request types for any service type to its default, you can use the following clause:

```
METHODS 'DEFAULT'
```

SECURE clause Specifies whether the service should be accessible on a secure or non-secure listener. ON indicates that only HTTPS connections are accepted, and that connections received on the HTTP port are automatically redirected to the HTTPS port. OFF indicates that both HTTP and HTTPS connections are accepted, provided that the necessary ports are specified when

starting the web server. The default value is OFF.

USER clause Specifies a database user, or group of users, with privileges to execute the web service request. A USER clause must be specified when the service is configured with AUTHORIZATION OFF and should be specified with AUTHORIZATION ON (the default). An HTTP request made to a service requiring authorization results in a [401 Authorization Required](#) HTTP response status. Based on this response, the web browser prompts for a user ID and password.

Caution

It is strongly recommended that you specify a USER clause when authorization is enabled (the default). Otherwise, authorization is granted to all users.

The USER clause controls which database user accounts can be used to process service requests. Database access permissions are restricted to the privileges assigned to the user of the service.

statement Specifies a command, such as a stored procedure call, to invoke when the service is accessed.

A DISH service is the only service that must either define a null statement, or not define a statement. A SOAP service must define a statement. Any other SERVICE can have a NULL statement, but only if it is configured with AUTHORIZATION ON.

An HTTP request to a non-DISH service with no *statement* specifies the SQL expression to execute within its URL. Although authorization is required, this capability should not be used in production systems because it makes the server vulnerable to SQL injections. When a statement is defined within the service, the specified SQL statement is the only statement that can be executed through the service.

In a typical web service application, you use *statement* to call a function or procedure. You can pass host variables as parameters to access client-supplied HTTP variables.

The following *statement* demonstrates a procedure call that passes two host variables to a procedure named AuthenticateUser. This call presumes that a web client supplies the user_name and user_password variables:

```
CALL AuthenticateUser ( :user_name, :user_password );
```

Remarks

Service definitions are stored within the ISYSWEBSERVICE table and can be examined from the SYSWEBSERVICE view.

Privileges

You must have the MANAGE ANY WEB SERVICE system privilege.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.
- **Transact-SQL** CREATE SERVICE is supported by Adaptive Server Enterprise for SOAP types

only.

Related Information

[How to access client-supplied HTTP variables and headers](#)

Variables and headers in an HTTP client request can be accessed by web services.

[SOAP data types](#)

By default, the XML encoding of parameter input is string and the result set output for SOAP service formats contains no information that specifically describes the data type of the columns in the result set. For all formats, parameter data types are string.

[The database server as an HTTP web server](#)

The database server contains a built-in HTTP web server that allows you to provide online web services from a database.

[How to create and customize a root web service](#)

An HTTP client request that does not match any web service request is processed by the **root** web service if a **root** web service is defined.

[Tutorial: Using a database server to access a SOAP/DISH service](#)

This tutorial illustrates how to create a SOAP server that converts a web client-supplied Fahrenheit temperature value to Celsius.

[ALTER SERVICE statement \[SOAP web service\]](#)

Alters an existing SOAP or DISH service over HTTP.

[DROP SERVICE statement](#)

Drops a web service.

[SYSWEBSERVICE system view](#)

Each row in the SYSWEBSERVICE system view holds a description of a web service. The underlying system table for this view is ISYSWEBSERVICE.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

CREATE SPATIAL REFERENCE SYSTEM statement

Creates or replaces a spatial reference system.

Syntax

```
{ CREATE [ OR REPLACE ] SPATIAL REFERENCE SYSTEM
| CREATE SPATIAL REFERENCE SYSTEM IF NOT EXISTS }
srs-name
[ srs-attribute ] [ srs-attribute ... ]

srs-name : string

srs-attribute :
IDENTIFIED BY srs-id
| DEFINITION { definition-string | NULL }
| ORGANIZATION { organization-name IDENTIFIED BY organization-srs-id | NULL }
| TRANSFORM DEFINITION { transform-definition-string | NULL }
| LINEAR UNIT OF MEASURE linear-unit-name
| ANGULAR UNIT OF MEASURE { angular-unit-name | NULL }
| TYPE { ROUND EARTH | PLANAR }
| COORDINATE coordinate-name { UNBOUNDED | BETWEEN low-number AND
high-number }
| ELLIPSOID SEMI MAJOR AXIS semi-major-axis-length { SEMI MINOR AXIS semi-minor-
axis-length | INVERSE FLATTENING inverse-flattening-ratio }
| SNAP TO GRID { grid-size | DEFAULT }
| TOLERANCE { tolerance-distance | DEFAULT }
| AXIS ORDER axis-order
| POLYGON FORMAT polygon-format
| STORAGE FORMAT storage-format

srs-id : integer

semi-major-axis-length : number

semi-minor-axis-length : number

inverse-flattening-ratio : number

grid-size : DOUBLE : usually between 0 and 1

tolerance-distance : number

axis-order : { 'x/y/z/m' | 'long/lat/z/m' | 'lat/long/z/m' }

polygon-format : { 'CounterClockWise' | 'Clockwise' | 'EvenOdd' }

storage-format : { 'Internal' | 'Original' | 'Mixed' }
```

Parameters

OR REPLACE clause Specifying OR REPLACE creates the spatial reference system if it does not already exist in the database, and replaces it if it does exist. An error is returned if you attempt to replace a spatial reference system while it is in use. An error is also returned if you attempt to replace a spatial reference system that already exists in the database without specifying the OR REPLACE clause.

CREATE SPATIAL REFERENCE IF NOT EXISTS Specifying CREATE SPATIAL REFERENCE IF NOT EXISTS checks to see if a spatial reference system by that name already exists. If it does not exist, the database server creates the spatial reference system. If it does exist, no further action is performed and no error is returned.

IDENTIFIED BY clause Use this clause to specify the SRID (*srs-id*) for the spatial reference system. If the spatial reference system is defined by an organization with an *organization-srs-id*, then *srs-id* should be set to that value.

If the IDENTIFIED BY clause is not specified, then the SRID defaults to the *organization-srs-id* defined by either the ORGANIZATION clause or the DEFINITION clause. If neither clause defines an *organization-srs-id* that could be used as a default SRID, an error is returned.

When the spatial reference system is based on a well known coordinate system, but has a different geodesic interpretation, set the *srs-id* value to be 1000000000 (one billion) plus the well known value. For example, the SRID for a planar interpretation of the geodetic spatial reference system WGS 84 (ID 4326) would be 1000004326.

With the exception of SRID 0, spatial reference systems provided by SQL Anywhere that are not based on well known systems are given a SRID of 2000000000 (two billion) and above. The range of SRID values from 2000000000 to 2147483647 is reserved by SQL Anywhere and you should not create SRIDs in this range.

To reduce the possibility of choosing a SRID that is reserved by a defining authority such as OGC or by other vendors, you should not choose a SRID in the range 0 - 32767 (reserved by EPSG), or in the range 2147483547 - 2147483647.

Also, since the SRID is stored as a signed 32-bit integer, the number cannot exceed $2^{31}-1$ or 2147483647.

DEFINITION clause Use this clause to set, or override, default coordinate system settings. If any attribute is set in a clause other than the DEFINITION clause, it takes the value specified in the other clause regardless of what is specified in the DEFINITION clause.

definition-string is a string in the Spatial Reference System Well Known Text syntax as defined by SQL/MM and OGC. For example, the following query returns the definition for WGS 84.

```
SELECT ST_SpatialRefSys::ST_FormatWKT( definition )
FROM ST_SPATIAL_REFERENCE_SYSTEMS
WHERE srs_id=4326;
```

In Interactive SQL, if you double-click the value returned, an easier to read version of the value appears.

When the DEFINITION clause is specified, *definition-string* is parsed and used to choose default values for attributes. For example, *definition-string* may contain an AUTHORITY element that defines the *organization-name* and *organization-srs-id*.

Parameter values in *definition-string* are overridden by values explicitly set using the SQL statement clauses. For example, if the ORGANIZATION clause is specified, it overrides the value for ORGANIZATION in *definition-string*.

ORGANIZATION clause Use this clause to specify information about the organization that created the spatial reference system that the new spatial reference system is based on. *organization-name* is the name of the organization that created it; *organization-srs-id* is the numeric identifier the organization uses to identify the spatial reference system.

TRANSFORM DEFINITION clause Use this clause to specify a description of the transform to use for the spatial reference system. Currently, only the PROJ.4 transform is supported. For example, the *transform-definition-string* for WGS 84 is `'+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs'`.

If you specify an unsupported transform definition, an error is returned.

The transform definition is used by the ST_Transform method when transforming data between spatial reference systems. Some transforms may still be possible even if there is no *transform-definition-string* defined.

LINEAR UNIT OF MEASURE clause Use this clause to specify the linear unit of measure for the spatial reference system. The value you specify must match a linear unit of measure that is defined in the ST_UNITS_OF_MEASURE consolidated view.

If this clause is not specified, and is not defined in the DEFINITION clause, the default is METRE.

To add predefined units of measure to the database, use the sa_install_feature system procedure.

To add custom units of measure to the database, use the CREATE SPATIAL UNIT OF MEASURE statement.

Note While both METRE and METER are accepted spellings, METRE is preferred as it conforms to the SQL/MM standard.

ANGULAR UNIT OF MEASURE clause Use this clause to specify the angular unit of measure for the spatial reference system. The value you specify must match an angular unit of measure defined in the ST_UNITS_OF_MEASURE consolidated view.

If this clause is not specified, and is not defined in the DEFINITION clause, the default is DEGREE for geographic spatial reference systems and NULL for non-geographic spatial reference systems.

The angular unit of measure must be non-NULL for geographic spatial reference systems and it must be NULL for non-geographic spatial reference systems.

To add predefined units of measure to the database, use the sa_install_feature system procedure.

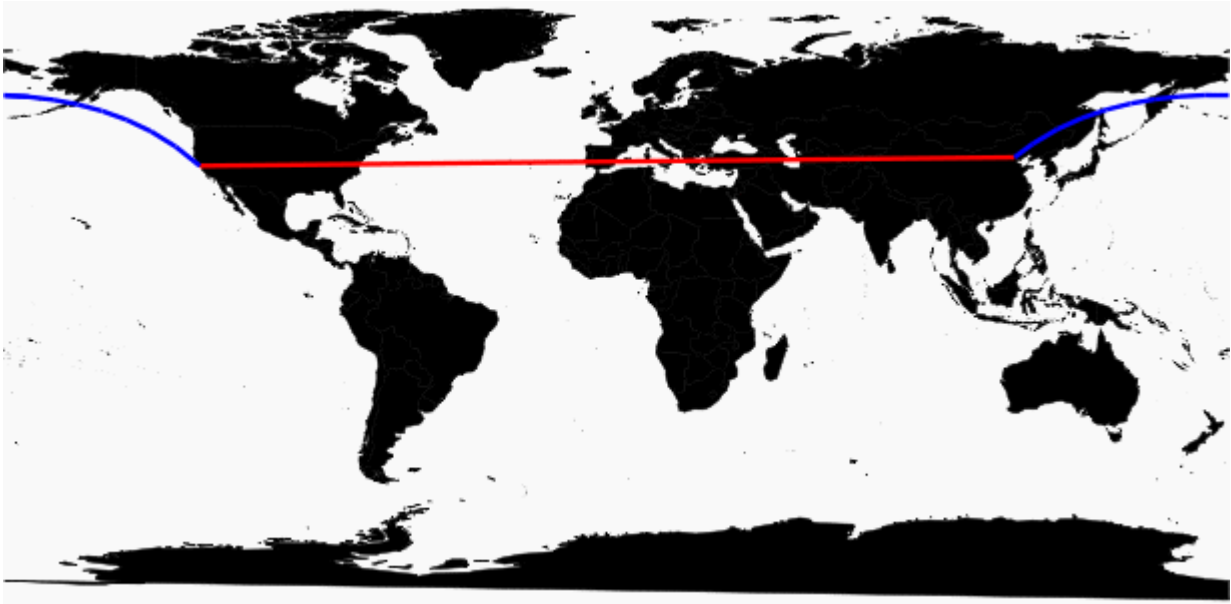
To add custom units of measure to the database, use the CREATE SPATIAL UNIT OF MEASURE statement.

TYPE clause Use the TYPE clause to control how the SRS interprets lines between points. For geographic spatial reference systems, the TYPE clause can specify either ROUND EARTH (the default) or PLANAR. The ROUND EARTH model interprets lines between points as great elliptic arcs. Given two points on the surface of the Earth, a plane is selected that intersects the two points and the center of the Earth. This plane intersects the Earth, and the line between the two points is the shortest distance along this intersection.

For two points that lie directly opposite each other, there is not a single unique plane that intersects the two points and the center of the Earth. Line segments connecting these antipodal points are not valid and give an error in the ROUND EARTH model.

The ROUND EARTH model treats the Earth as a spheroid and selects lines that follow the curvature of the Earth. In some cases, it may be necessary to use a planar model where a line between two points is interpreted as a straight line in the equirectangular projection where $x=\text{long}$, $y=\text{lat}$.

In the following example, the blue line shows the line interpretation used in the ROUND EARTH model and the red line shows the corresponding PLANAR model.



The PLANAR model may be used to match spatial interpretation used in other software products. The PLANAR model may also be useful because there are some limitations for methods that are not supported in the ROUND EARTH model (such as `ST_Area`, `ST_ConvexHull`) and some are partially supported (`ST_Distance` only supported between point geometries). Geometries based on circularstrings are not supported in ROUND EARTH spatial reference systems.

For non-geographic SRSs, the type must be PLANAR (and that is the default if the TYPE clause is not specified and either the DEFINITION clause is not specified or it uses a non-geographic definition).

COORDINATE clause Use this clause to specify the bounds on the spatial reference system's dimensions. *coordinate-name* is the name of the coordinate system used by the spatial reference system. For non-geographic coordinate systems, *coordinate-name* can be `x`, `y`, or `m`. For geographic coordinate systems, *coordinate-name* can be `LATITUDE`, `LONGITUDE`, `z`, or `m`.

Specify `UNBOUNDED` to place no bounds on the dimensions. Use the `BETWEEN` clause to set low and high bounds.

The X and Y coordinates must have associated bounds. For geographic spatial reference systems, the longitude coordinate is bounded between -180 and 180 degrees and the latitude coordinate is bounded between -90 and 90 degrees by default the unless `COORDINATE` clause overrides these settings. For non-geographic spatial reference systems, the `CREATE` statement must specify bounds for both X and Y coordinates.

`LATITUDE` and `LONGITUDE` are used for geographic coordinate systems. The bounds for `LATITUDE` and `LONGITUDE` default to the entire Earth, if not specified.

ELLIPSOID clause Use the ellipsoid clause to specify the values to use for representing the Earth as an ellipsoid for spatial reference systems of type ROUND EARTH. If the DEFINITION clause is present, it can specify ellipsoid definition. If the ELLIPSOID clause is specified, it overrides this default ellipsoid.

The Earth is not a perfect sphere because the rotation of the Earth causes a flattening so that the distance from the center of the Earth to the North or South pole is less than the distance from the center to the equator. For this reason, the Earth is modeled as an ellipsoid with different values for the semi-major axis (distance from center to equator) and semi-minor axis (distance from center to the pole). It is most common to define an ellipsoid using the semi-major axis and the inverse flattening, but it can instead be specified using the semi-minor axis (for example, this approach must be used when a perfect sphere is used to approximate the Earth). The semi-major and semi-minor axes are defined in the linear units of the spatial reference system, and the inverse flattening (1/f) is a ratio:

$$1/f = (\text{semi-major-axis}) / (\text{semi-major-axis} - \text{semi-minor-axis})$$

SQL Anywhere uses the ellipsoid definition when computing distance in geographic spatial reference systems.

The ellipsoid must be defined for geographic spatial reference systems (either in the DEFINITION clause or the ELLIPSOID clause), and it must not be specified for non-geographic spatial reference systems.

SNAP TO GRID clause For flat-Earth (planar) spatial reference systems, use the SNAP TO GRID clause to define the size of the grid SQL Anywhere uses when performing calculations. By default, SQL Anywhere selects a grid size so that 12 significant digits can be stored at all points in the space bounds for X and Y. For example, if a spatial reference system bounds X between -180 and 180 and Y between -90 and 90, then a grid size of 0.000000001 (1E-9) is selected.

grid-size must be large enough so that points snapped to the grid can be represented with equal precision at all points in the bounded space. If *grid-size* is too small, the server reports an error.

When set to 0, no snapping to grid is performed.

For round-Earth spatial reference systems, SNAP TO GRID must be set to 0.

Specify SNAP TO GRID DEFAULT to set the grid size to the default that the database server would use.

TOLERANCE clause For flat-Earth (planar) spatial reference systems, use the TOLERANCE clause to specify the precision to use when comparing points. If the distance between two points is less than *tolerance-distance*, the two points are considered equal. Setting *tolerance-distance* allows you to control the tolerance for imprecision in the input data or limited internal precision. By default, *tolerance-distance* is set to be equal to *grid-size*.

When set to 0, two points must be exactly equal to be considered equal.

For round-Earth spatial reference systems, TOLERANCE must be set to 0.

POLYGON FORMAT clause Internally, SQL Anywhere interprets polygons by looking at the orientation of the constituent rings. As one travels a ring in the order of the defined points, the inside of the polygon is on the left side of the ring. The same rules are applied in PLANAR and ROUND EARTH spatial reference systems.

The interpretation used by SQL Anywhere is a common but not universal interpretation. Some products use the exact opposite orientation, and some products do not rely on ring orientation to

interpret polygons. The POLYGON FORMAT clause can be used to select a polygon interpretation that matches the input data, as needed. The following values are supported:

- o **'CounterClockwise'** The input follows SQL Anywhere's internal interpretation: the inside of the polygon is on the left side while following ring orientation.
- o **'Clockwise'** The input follows the opposite of SQL Anywhere's approach: the inside of the polygon is on the right side while following ring orientation.
- o **'EvenOdd'** EvenOdd is the default format. With EvenOdd, the orientation of rings is ignored and the inside of the polygon is instead determined by looking at the nesting of the rings, with the exterior ring being the largest ring and interior rings being smaller rings inside this ring. A ray is traced from a point within the rings and radiating outward crossing all rings. If the number the ring being crossed is an even number, it is an outer ring. If it is odd, it is an inner ring.

STORAGE FORMAT clause When you insert spatial data into the database from an external format (such as WKT or WKB), the database server normalizes the data to improve the performance and semantics of spatial operations. The normalized representation may differ from the original representation (for example, in the orientation of polygon rings or the precision stored in individual coordinates). While spatial equality is maintained after the normalization, some original input characteristics may not be reproducible, such as precision and ring orientation. In some cases you may want to store the original representation, either exclusively, or in addition to the normalized representation.

To control what is stored, specify the STORAGE FORMAT clause followed by one of the following values:

- o **'Internal'** SQL Anywhere stores only the normalized representation. Specify this value when the original input characteristics do not need to be reproduced. This is the default for planar spatial reference systems (TYPE PLANAR).

Note If you are using MobiLink to synchronize your spatial data, you should specify **Mixed**. MobiLink tests for equality during synchronization, which requires the data in its original format.

- o **'Original'** SQL Anywhere stores only the original representation. The original input characteristics can be reproduced, but all operations on the stored values must repeat normalization steps, possibly slowing down operations on the data.
- o **'Mixed'** SQL Anywhere stores the internal version and, if it is different from the original version, it stores the original version as well. By storing both versions, the original representation characteristics can be reproduced and operations on stored values do not need to repeat normalization steps. However, storage requirements may increase significantly because potentially two representations are being stored for each geometry. Mixed is the default format for round-Earth spatial reference systems (TYPE ROUND EARTH).

Remarks

For a geographic spatial reference system, you can specify both a LINEAR *and* an ANGULAR unit of measure; otherwise for non-geographic, you specify only a LINEAR unit of measure. The LINEAR unit of measure is used for computing distance between points and areas. The ANGULAR unit of measure tells how the angular latitude/longitude are interpreted and is NULL for projected coordinate systems, non-NULL for geographic coordinate systems.

All derived geometries returned by operations are normalized.

When working with data that is being synchronized with a non-SQL Anywhere database, STORAGE FORMAT should be set to either 'Original' or 'Mixed' so that the original characteristics of the data can be preserved.

If you use this statement in a procedure, do not specify the password (IDENTIFIED BY clause) as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

Privileges

You must have the MANAGE ANY SPATIAL OBJECT or CREATE ANY OBJECT system privilege.

Side effects

None

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example creates a spatial reference system named mySpatialRS:

```

CREATE SPATIAL REFERENCE SYSTEM "mySpatialRS"
IDENTIFIED BY 1000026980
LINEAR UNIT OF MEASURE "metre"
TYPE PLANAR
COORDINATE X BETWEEN 171266.736269555 AND 831044.757769222
COORDINATE Y BETWEEN 524881.608973277 AND 691571.125115319
DEFINITION 'PROJCS["NAD83 / Kentucky South",
GEOGCS["NAD83",
DATUM["North_American_Datum_1983",
SPHEROID["GRS 1980",6378137,298.257222101,AUTHORITY["EPSG","7019"]],
AUTHORITY["EPSG","6269"]],
PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],
UNIT["degree",0.01745329251994328,AUTHORITY["EPSG","9122"]],
AUTHORITY["EPSG","4269"]],
UNIT["metre",1,AUTHORITY["EPSG","9001"]],
PROJECTION["Lambert_Conformal_Conic_2SP"],
PARAMETER["standard_parallel_1",37.93333333333333],
PARAMETER["standard_parallel_2",36.73333333333333],
PARAMETER["latitude_of_origin",36.33333333333334],
PARAMETER["central_meridian",-85.75],
PARAMETER["false_easting",500000],
PARAMETER["false_northing",500000],
AUTHORITY["EPSG","26980"],
AXIS["X",EAST],
AXIS["Y",NORTH]]'
TRANSFORM DEFINITION '+proj=lcc
+lat_1=37.93333333333333+lat_2=36.73333333333333+lat_0=36.33333333333334+lon_0=-
85.75+x_0=500000+y_0=500000+ellps=GRS80+datum=NAD83+units=m+no_defs';

```

Related Information

[Spatial reference systems \(SRS\) and Spatial reference identifiers \(SRID\)](#)

In the context of spatial databases, the defined space in which geometries are described is called a **spatial reference system (SRS)**.

[Spatial data](#)

Spatial data is data that describes the position, shape, and orientation of objects in a defined space.

[sa_install_feature system procedure](#)

Installs additional features, for example additional spatial features.

[CREATE SPATIAL UNIT OF MEASURE statement](#)

Creates or replaces a spatial unit of measurement.

[ST_UNITS_OF_MEASURE consolidated view](#)

Each row of the ST_UNITS_OF_MEASURE system view describes a unit of measure defined in the database. This view offers more information than the SYSUNITOFMEASURE system view.

[ST_SPATIAL_REFERENCE_SYSTEMS consolidated view](#)

Each row of the ST_SPATIAL_REFERENCE_SYSTEMS system view describes an SRS defined in the database. This view offers a slightly different amount of information than the SYSSPATIALREFERENCESYSTEM system view.

[ALTER SPATIAL REFERENCE SYSTEM statement](#)

Changes the settings of an existing spatial reference system. See the Remarks section for considerations before altering a spatial reference system.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CREATE SPATIAL UNIT OF MEASURE statement

Creates or replaces a spatial unit of measurement.

Syntax

```
CREATE [ OR REPLACE ] SPATIAL UNIT OF MEASURE identifier  
TYPE { LINEAR | ANGULAR }  
[ CONVERT USING number ]
```

Parameters

OR REPLACE clause Including the OR REPLACE creates a new spatial unit of measure, or replaces an existing spatial unit of measure with the same name. This clause preserves existing privileges. An error is returned if you attempt to replace a spatial unit that is already in use.

TYPE clause Defines whether the unit of measure is used for angles (ANGULAR) or distances (LINEAR).

CONVERT USING The conversion factor for the spatial unit relative to the base unit. For linear units, the base unit is METRE. For angular units, the base unit is RADIAN.

Remarks

The CONVERT USING clause is used to define how to convert a measurement in the defined unit of measure to the base unit of measure (radians or meters). The measurement is multiplied by the supplied conversion factor to get a value in the base unit of measure. For example, a measurement of 512 millimeters would be multiplied by a conversion factor of 0.001 to get a measurement of 0.512 metres.

Spatial reference systems always include a linear unit of measure to be used when calculating distances (ST_Distance or ST_Length), or area. For example, if the linear unit of measure for a spatial reference system is miles, then the area unit used is square miles. In some cases, spatial methods accept an optional parameter that specifies the linear unit of measure to use. For example, if the linear unit of measure for a spatial reference system is in miles, you could retrieve the distance between two geometries in meters by using the optional parameter 'metre'.

For projected coordinate systems, the X and Y coordinates are specified in the linear unit of the spatial reference system. For geographic coordinate systems, the latitude and longitude are specified in the angular units of measure associated with the spatial reference system. In many cases, this angular unit of measure is degrees but any valid angular unit of measure can be used.

You can use the sa_install_feature system procedure to add predefined units of measure to your database.

Privileges

You must have the MANAGE ANY SPATIAL OBJECT or CREATE ANY OBJECT system privilege.

Side effects

None

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example creates a spatial unit of measure named Test.

```
CREATE SPATIAL UNIT OF MEASURE Test  
TYPE LINEAR  
CONVERT USING 15;
```

Related Information

[Spatial data](#)

Spatial data is data that describes the position, shape, and orientation of objects in a defined space.

[sa_install_feature system procedure](#)

Installs additional features, for example additional spatial features.

[DROP SPATIAL UNIT OF MEASURE statement](#)

Drops a spatial unit of measurement.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CREATE STATISTICS statement

Recreates the column statistics used by the optimizer, and stores them in the ISYSCOLSTAT system table.

Syntax

CREATE STATISTICS *object-name* [(*column-list*)]

object-name :

table-name | *materialized-view-name* | *temp-table-name*

Remarks

The CREATE STATISTICS statement recreates the column statistics that the database server uses to optimize database queries, and can be performed on base tables, materialized views, local temporary tables, and global temporary tables. You cannot create statistics on proxy tables. Column statistics include histograms, which reflect the distribution of data in the database for the specified columns. By default, column statistics are automatically created for tables with five or more rows.

In rare circumstances, when your database queries are very variable, and when data distribution is not uniform or the data is changing frequently, you can improve performance by executing the CREATE STATISTICS statement on a table or column.

When executing, the CREATE STATISTICS statement updates existing column statistics regardless of the size of the table, unless the table is empty, in which case nothing is done. If column statistics exist for an empty table, they remain unchanged by the CREATE STATISTICS statement. To remove column statistics for an empty table, execute the DROP STATISTICS statement.

The process of executing CREATE STATISTICS performs a complete scan of the table. For this reason, careful consideration should be made before executing a CREATE STATISTICS statement.

If you drop statistics, it is recommended that you recreate them using the CREATE STATISTICS statement. Without statistics, the optimizer can generate inefficient data access plans, causing poor database performance.

Privileges

You must be the table owner, or have the MANAGE ANY STATISTICS or CREATE ANY OBJECT system privilege.

Side effects

Execution plans may change.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statement updates the column statistics for the ProductID column of the SalesOrderItems table:

```
CREATE STATISTICS GROUP0.SalesOrderItems ( ProductID );
```

Related Information

[Optimizer estimates and statistics](#)

The optimizer chooses a strategy for processing a statement based on **column statistics** stored in the database and on **heuristics**.

[DROP STATISTICS statement](#)

Erases all column statistics on the specified columns.

[LOAD TABLE statement](#)

Imports bulk data into a database table from an external file.

[SYSCOLSTAT system view](#)

The SYSCOLSTAT system view contains the column statistics, including histograms, that are used by the optimizer. The contents of this view are best retrieved using the `sa_get_histogram` stored procedure or the Histogram utility. The underlying system table for this view is ISYSCOLSTAT.

[Histogram utility \(dbhist\)](#)

Converts a histogram into a Microsoft Excel chart containing information about the selectivity of predicates.

[sa_get_histogram system procedure](#)

Retrieves the histogram for a column.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CREATE SUBSCRIPTION statement [SQL Remote]

Creates a subscription for a user to a publication.

Syntax

```
CREATE SUBSCRIPTION  
TO publication-name [ ( subscription-value ) ]  
FOR subscriber-id
```

publication-name : *identifier*

subscription-value : *string*

subscriber-id : *string*

Parameters

publication-name The name of the publication to which the user is being subscribed. This can include the owner of the publication.

subscription-value A string that is compared to the subscription expression of the publication. The subscriber receives all rows for which the subscription expression matches the subscription value.

You can specify a variable name for *subscription-value*. If the variable is NULL, the value behaves as if an empty string was specified.

subscriber-id The user ID of the subscriber to the publication. At the consolidated database, when you create a subscription to a remote user, the remote user must have been granted REMOTE privilege. At the remote database, when you create a subscription to the consolidated user, that user must have been granted CONSOLIDATED privilege.

Remarks

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 user to a publication on the remote database. By default, the Extraction utility (dbxtract) and the **Extract Database Wizard** grant the appropriate PUBLISH and CONSOLIDATE privilege to users in the remote databases.

If *subscription-value* is supplied, 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.

Note For *required* parameters that accept variable names, an error is returned if one of the following conditions is true:

- The variable does not exist
- The contents of the variable are NULL
- The variable exceeds the length allowed by the parameter
- The data type of the variable does not match that required by the parameter

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statement creates a subscription for the user p_chin to the publication pub_sales. The subscriber receives all rows for which the subscription expression has a value of Eastern.

```
CREATE SUBSCRIPTION
TO pub_sales ( 'Eastern' )
FOR p_chin;
```

The following statement creates a subscription for user name Sam_Singer to the CustomerPub publication, which was created using a WHERE clause:

```
CREATE SUBSCRIPTION
TO CustomerPub
FOR Sam_Singer;
```

The following statement creates a subscription for user name Sam_Singer to the PubOrders publication, defined with a subscription expression SalesRepresentative, requesting the rows for Sam Singer's own sales:

```
CREATE SUBSCRIPTION
TO PubOrders ( '856' )
FOR Sam_Singer;
```

Example

The following statement creates a variable for the value 'ny':

```
CREATE VARIABLE @state LONG VARCHAR ;
SET @state = 'ny' ;
```

The following statement creates a publication called pub_customer:

```
CREATE PUBLICATION pub_customer (
TABLE DBA>Customer ( ID, company_name, City, State )
SUBSCRIBE BY State ) ;
```

The following statement creates a subscription for user name Sam_Singer

```
CREATE SUBSCRIPTION  
  TO pub_customer ( @dan )  
  FOR Sam_Singer ;
```

The subscriber receives all rows for which the subscription expression has a value of the variable @dan, which is set to NY state.

Related Information

[Subscriptions](#)

You subscribe a user to a publication by creating a **subscription**. Each database that shares information in a publication must have a subscription to the publication.

[Publishing only some rows in a table \(SQL Central\)](#)

To create a publication that contains only some of the rows in a table, you must write a search condition that matches only the rows you want to publish.

[Replicating the primary key pool \(SQL\)](#)

Create a separate publication for the primary key pool and subscribe users to it.

[Publishing only some rows using the SUBSCRIBE BY clause \(SQL Central\)](#)

Create a publication using the SUBSCRIBE BY clause.

[Publishing only some rows using a WHERE clause \(SQL Central\)](#)

Create a publication that uses a WHERE clause to include all the columns, but only some of the rows of a table.

[DROP SUBSCRIPTION statement \[SQL Remote\]](#)

Drops a subscription for a user from a publication.

[GRANT REMOTE statement \[SQL Remote\]](#)

Identifies a database immediately below the current database in a SQL Remote hierarchy, that will receive messages from the current database. These are called remote users.

[GRANT CONSOLIDATE statement \[SQL Remote\]](#)

Identifies the database immediately above the current database in a SQL Remote hierarchy, that will receive messages from the current database.

[SYNCHRONIZE SUBSCRIPTION statement \[SQL Remote\]](#)

Synchronizes a subscription for a user to a publication.

[START SUBSCRIPTION statement \[SQL Remote\]](#)

Starts a subscription for a user to a publication.

[GRANT PUBLISH statement \[SQL Remote\]](#)

Grants publisher rights to a user ID. You must have the SYS_REPLICATION_ADMIN_ROLE system role to grant publisher rights.

[SYSSUBSCRIPTION system view](#)

Each row in the SYSSUBSCRIPTION system view describes a subscription from one user ID (which must have the REMOTE system privilege) to one publication. The underlying system table for this view is ISYSSUBSCRIPTION.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CREATE SYNCHRONIZATION PROFILE statement [MobiLink]

Creates a SQL Anywhere synchronization profile.

Syntax

```
CREATE [ OR REPLACE ] SYNCHRONIZATION PROFILE name string
```

Parameters

OR REPLACE clause Specifying CREATE OR REPLACE SYNCHRONIZATION PROFILE replaces the definition of the named synchronization profile if it already exists.

name Specifies the name of the synchronization profile to create. Each profile must have a unique name.

string Specify a valid option string as described below. Option strings are specified as semicolon delimited lists of elements of the form *option-name=option-value*. For example `subscription=s1;verbosity=high`.

Remarks

Synchronization profiles are named collections of synchronization options that can be used to control synchronization.

For options that take a Boolean value, setting the value to TRUE is equivalent to specifying the corresponding option on the command line.

The following values can be used to specify TRUE: TRUE, ON, 1, YES.

The following values can be used to specify FALSE: FALSE, OFF, 0, NO.

When setting extended options, use the following syntax:

```
CREATE SYNCHRONIZATION PROFILE myprofile 's=mysub;e={ctp=tcip;  
adr=' 'host=localhost;port=2439' '}'
```

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Related Information

[MobiLink synchronization profiles](#)

Synchronization profiles allow you to place some dbmlsync options in the database. The synchronization profile you create can contain a variety of synchronizations options.

[ALTER SYNCHRONIZATION PROFILE statement \[MobiLink\]](#)

Changes a SQL Anywhere synchronization profile. Synchronization profiles are named collections of synchronization options that can be used to control synchronization.

[DROP SYNCHRONIZATION PROFILE statement \[MobiLink\]](#)

Deletes a SQL Anywhere synchronization profile.

[SYNCHRONIZE statement \[MobiLink\]](#)

Synchronizes a SQL Anywhere database with a MobiLink server. The synchronization options can be specified in the statement itself.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]

Creates a subscription in a SQL Anywhere remote database between a MobiLink user and a publication.

Syntax

```
CREATE SYNCHRONIZATION SUBSCRIPTION[ subscription-name ]  
TO publication-name  
[ FOR ml-username, ... ]  
[ TYPE network-protocol ]  
[ ADDRESS protocol-options ]  
[ OPTION option=value, ... ]  
[ SCRIPT VERSION script-version ]
```

subscription-name : *identifier*

ml-username : *identifier*

network-protocol : **http** | **https** | **tls** | **tcPIP**

protocol-options : *string*

value : *string* | *integer*

script-version : *string*

Parameters

subscription-name A unique name that you can use to identify this subscription. It is strongly recommended that you name all your subscriptions.

TO clause This clause specifies the name of a publication.

FOR clause This clause specifies one or more MobiLink user names. If you specify more than one user name, a separate subscription is created for each user. If you specify a subscription name, only one MobiLink user name can be specified.

ml-username is a user who is authorized to synchronize with the MobiLink server.

Omit the FOR clause to set the protocol type, protocol options, and extended options for a publication. If the FOR clause is omitted, you cannot specify a subscription name or use the SCRIPT VERSION clause.

TYPE clause This clause specifies the network protocol to use for synchronization. The default protocol is tcPIP.

ADDRESS clause This clause specifies network protocol options such as the location of the MobiLink server. Multiple options must be separated by semicolons.

OPTION clause This clause allows you to set extended options for the subscription. If no FOR clause is provided, the extended options act as default settings for the publication.

SCRIPT VERSION clause This clause specifies the script version to use during synchronization. Typically, you must specify a new script version for each schema change you

implement.

You cannot use the `SCRIPT VERSION` clause if the `FOR` clause is omitted.

Remarks

If no *subscription-name* is specified, a unique name is generated. The generated subscription name is the same as the publication name, provided it is unique. Otherwise, a unique name is formed by adding a number to the end of the publication name, for example, `pub001`, `pub002`, and so on.

The *network-protocol*, *protocol-options*, and *option* values can be set in several places.

This statement causes options and other information to be stored in the `ISYSSYNC` system table. Anyone with privileges to view data in the `SYSSYNC` system view can view the information, which could include passwords and encryption certificates. To avoid this potential security issue, you can specify the information on the `dbmlsync` command line.

You must have exclusive access to all tables referenced in the publication to execute the statement.

Privileges

You must have the `SYS_REPLICATION_ADMIN_ROLE` system role.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example creates a subscription named `sales` between the MobiLink user `SSinger` and the publication called `sales_publication` owned by *user*. When the subscription is synchronized, the script version `sales_v1` is used and tables are locked in exclusive mode:

```
CREATE SYNCHRONIZATION SUBSCRIPTION sales
TO user.sales_publication
FOR SSinger
OPTION locktables='exclusive'
SCRIPT VERSION 'sales_v1'
```

The following example omits the `FOR` clause and stores settings for the publication called `sales_publication`:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
TO user.sales_publication
ADDRESS 'host=test.internal;port=2439;
OPTION locktables=exclusive';
```

Related Information

[Synchronization subscription creation](#)

After creating MobiLink users and publications, you must subscribe at least one MobiLink user to one or more pre-existing publications. You do this by creating synchronization subscriptions.

[MobiLink users in a synchronization system](#)

A **MobiLink user**, also called a **synchronization user**, is the name you use to authenticate when you connect to the MobiLink server.

[MobiLink SQL Anywhere client extended options](#)

Extended options can be specified on the dbmlsync command line using the -e or -eu options, or they can be stored in the database.

[How dbmlsync resolves conflicting options](#)

Dbmlsync combines options stored in the database with those specified on the command line. If conflicting options are specified, dbmlsync resolves them as follows. In the following list, options specified by methods occurring earlier in the list take precedence over those occurring later in the list.

[Script versions](#)

Scripts are organized into groups called **script versions**. By specifying a particular script version, MobiLink clients can select which set of synchronization scripts are used to process the upload and prepare the download.

[ALTER SYNCHRONIZATION SUBSCRIPTION statement \[MobiLink\]](#)

Alters the properties of a synchronization subscription in a SQL Anywhere remote database.

[DROP SYNCHRONIZATION SUBSCRIPTION statement \[MobiLink\]](#)

Drops a synchronization subscription in a remote database.

[MobiLink client network protocol options](#)

MobiLink client utilities use the following MobiLink client network protocol options when connecting to the MobiLink server.

[CommunicationType \(ctp\) extended option](#)

Specifies the type of network protocol to use for connecting to the MobiLink server.

[SYSSYNC system view](#)

The SYSSYNC system view contains information relating to synchronization.

[MobiLink SQL Anywhere client utility \(dbmlsync\) syntax](#)

Use the dbmlsync utility to synchronize SQL Anywhere remote databases with a consolidated database. To run dbmlsync you must have the SYS_RUN_REPLICATION_ROLE system privilege.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CREATE SYNCHRONIZATION USER statement [MobiLink]

Creates a MobiLink user in a SQL Anywhere remote database.

Syntax

```
CREATE SYNCHRONIZATION USER ml-username  
[ TYPE network-protocol ]  
[ ADDRESS protocol-options ]  
[ OPTION option=value, ... ]
```

ml-username : *identifier*

network-protocol :

tcpip

| **http**

| **https**

| **tls**

protocol-options : *string*

value : *string* | *integer*

Parameters

ml_username A name identifying a MobiLink user.

TYPE clause This clause specifies the network protocol to use for synchronization. The default protocol is tcpip.

ADDRESS clause This clause specifies *protocol-options* in the form *keyword=value*, separated by semicolons. Which settings you supply depends on the communication protocol you are using (TCP/IP, TLS, HTTP, or HTTPS).

OPTION clause The OPTION clause allows you to set extended options using *option=value* in a comma-separated list.

The values for each option cannot contain equal signs or semicolons. The database server accepts any option that you enter without checking for its validity. Therefore, if you misspell an option or enter an invalid value, no error message appears until you run the dbmlsync command to perform synchronization.

Options set for a synchronization user can be overridden in individual subscriptions or on the dbmlsync command line.

The *network-protocol*, *protocol-options*, and *options* can be set in several places.

This statement causes options and other information to be stored in the SQL Anywhere ISYSSYNC system table. Anyone with the correct privileges to view the SYSSYNC system view can view the information, which could include passwords and encryption certificates. To avoid this potential security issue, you can specify the information on the dbmlsync command line.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example creates a MobiLink user named SSinger, who synchronizes over TCP/IP with a server computer named mlserver.mycompany.com using the password Sam. The use of a password in the user definition is *not* secure.

```
CREATE SYNCHRONIZATION USER SSinger
TYPE http
ADDRESS 'host=mlserver.mycompany.com'
OPTION MobiLinkPwd='Sam';
```

Related Information

[MobiLink client/server communications encryption](#)

You can encrypt MobiLink client/server communication using transport layer security.

[MobiLink users in a synchronization system](#)

A **MobiLink user**, also called a **synchronization user**, is the name you use to authenticate when you connect to the MobiLink server.

[MobiLink SQL Anywhere client extended options](#)

Extended options can be specified on the dbmlsync command line using the -e or -eu options, or they can be stored in the database.

[How dbmlsync resolves conflicting options](#)

Dbmlsync combines options stored in the database with those specified on the command line. If conflicting options are specified, dbmlsync resolves them as follows. In the following list, options specified by methods occurring earlier in the list take precedence over those occurring later in the list.

[ALTER SYNCHRONIZATION USER statement \[MobiLink\]](#)

Alters the properties of a MobiLink user in a SQL Anywhere remote database.

[DROP SYNCHRONIZATION USER statement \[MobiLink\]](#)

Drops one or more synchronization users from a SQL Anywhere remote database.

[CommunicationType \(ctp\) extended option](#)

Specifies the type of network protocol to use for connecting to the MobiLink server.

[SYSSYNC system view](#)

The SYSSYNC system view contains information relating to synchronization.

[MobiLink SQL Anywhere client utility \(dbmlsync\) syntax](#)

Use the dbmlsync utility to synchronize SQL Anywhere remote databases with a consolidated database. To run dbmlsync you must have the SYS_RUN_REPLICATION_ROLE system privilege.

[MobiLink client network protocol options](#)

MobiLink client utilities use the following MobiLink client network protocol options when connecting to the MobiLink server.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)» [Alphabetical list of SQL statements](#)

CREATE TABLE statement

Creates a new table in the database and, optionally, creates a table on a remote server.

Syntax

```

CREATE [ OR REPLACE ] [ GLOBAL TEMPORARY ] TABLE [ IF NOT EXISTS ] [
  owner.]table-name
( { column-definition | table-constraint | pctfree | like-clause },... ) like-clause | as-clause
[ { IN | ON } dbspace-name ]
[ ENCRYPTED ]
[ ON COMMIT { DELETE | PRESERVE } ROWS
  | NOT TRANSACTIONAL ]
[ AT location-string ]
[ SHARE BY ALL ]
[ WITH [ NO ] DATA ]

```

column-definition :

column-name data-type

```

[ COMPRESSED ]
[ INLINE { inline-length | USE DEFAULT } ]
[ PREFIX { prefix-length | USE DEFAULT } ]
[ [ NO ] INDEX ]
[ [ NOT ] NULL ]
[ DEFAULT default-value | IDENTITY ]
[ column-constraint ... ]
[ table-element ... ]

```

default-value :

special-value

```

| string
| global-variable
| [ - ] number
| ( constant-expression )
| ( sequence-expression )
| built-in-function( constant-expression )
| AUTOINCREMENT
| GLOBAL AUTOINCREMENT [ ( partition-size ) ]

```

*special-value :***CURRENT DATABASE**| **CURRENT DATE**| **CURRENT TIME**| [**CURRENT**] **TIMESTAMP**| **CURRENT PUBLISHER**| **CURRENT REMOTE USER**| [**CURRENT**] **USER**| [**CURRENT**] **UTC TIMESTAMP**| **LAST USER**| **NULL***column-constraint :*[**CONSTRAINT** *constraint-name*] {| **UNIQUE** [**CLUSTERED**]| **PRIMARY KEY** [**CLUSTERED**] [**ASC** | **DESC**]| **REFERENCES** *table-name* [(*column-name*)]| [**MATCH** [**UNIQUE**] { **SIMPLE** | **FULL** }]| [*action-list*] [**CLUSTERED**]| **CHECK** (*condition*)

| }

| **COMPUTE** (*expression*)*table-element :*{ *column-definition*| *table-constraint-definition*| *like-clause* }*table-constraint :*[**CONSTRAINT** *constraint-name*] {| **UNIQUE** [**CLUSTERED**] (*column-name* [**ASC** | **DESC**],...)| **PRIMARY KEY** [**CLUSTERED**] (*column-name* [**ASC** | **DESC**],...)| **CHECK** (*condition*)| *foreign-key-constraint*

| }

foreign-key-constraint :[**NOT NULL**] **FOREIGN KEY** [*role-name*]| [(*column-name* [**ASC** | **DESC**],...)]| **REFERENCES** *table-name*| [(*column-name*,...)]| [**MATCH** [**UNIQUE**] { **SIMPLE** | **FULL** }]| [*action-list*] [**CHECK ON COMMIT**] [**CLUSTERED**] [**FOR OLAP WORKLOAD**]*action-list :*[**ON UPDATE** *action*][**ON DELETE** *action*]

action :

CASCADE

| **SET NULL**

| **SET DEFAULT**

| **RESTRICT**

like-clause :

LIKE *source-table* [*like-option*,...]

like-option :

{ **INCLUDING** | **EXCLUDING** } *option* [,...]

option :

{ **IDENTITY**

| **DEFAULTS**

| **CONSTRAINTS**

| **INDEXES**

| **PRIMARY KEY**

| **FOREIGN KEYS**

| **COMMENTS**

| **STORAGE**

| **ALL** }

as-clause :

[(*column-name*,...)] **AS** (*select-statement*)

location-string :

remote-server-name.*[db-name]*.*[owner]*.*object-name*

| *remote-server-name*;*[db-name]*;*[owner]*;*object-name*

pctfree : **PCTFREE** *percent-free-space*

percent-free-space : *integer*

Parameters

LIKE clause The default for *like-option* is to exclude (EXCLUDING) all options. If *source-table* is a view, then *like-option* is ignored if it is not applicable.

Specify INCLUDE to include any of the following column attributes from the original table in the new table:

- **IDENTITY** Include default autoincrement information.
- **DEFAULTS** Include default value expressions, excluding autoincrement.
- **CONSTRAINTS** Include column and table check and unique constraints.
- **INDEXES** Include indexes, excluding primary and foreign keys.
- **PRIMARY KEY** Include the primary key.
- **FOREIGN KEY** Include foreign keys.
- **COMMENTS** Include comments on columns.
- **STORAGE** Include the compression setting.

AS clause Use this clause to clone a table based on a SELECT statement.

IN clause Use this clause to specify the dbspace in which the base table is located. If this clause is not specified, then the base table is created in the dbspace specified by the default_dbspace option.

Temporary tables can only be created in the temporary dbspace. If you are creating a GLOBAL TEMPORARY table, and specify IN, the table is created in the temporary dbspace. If you specify a user-defined dbspace, an error is returned.

ENCRYPTED clause The encrypted clause specifies that the table should be encrypted. You must enable table encryption when you create a database if you plan to encrypt tables. The table is encrypted using the encryption key and algorithm specified at database creation time.

ON COMMIT clause The ON COMMIT clause is allowed only for temporary tables. By default, the rows of a temporary table are deleted on COMMIT. If the SHARE BY ALL clause is specified, either ON COMMIT PRESERVE ROWS or NOT TRANSACTIONAL must be specified.

NOT TRANSACTIONAL clause The NOT TRANSACTIONAL clause is allowed when creating a global temporary table. A table created using NOT TRANSACTIONAL is not affected by either COMMIT or ROLLBACK. If the SHARE BY ALL clause is specified, either ON COMMIT PRESERVE ROWS or NOT TRANSACTIONAL must be specified.

AT clause Create a remote table on a different server specified by *location-string*, and a proxy table on the current database that maps to the remote table. The AT clause supports the semicolon (;) as a field delimiter in *location-string*. If no semicolon is present, a period is the field delimiter. This syntax allows file names and extensions to be used in the database and owner fields.

For example, the following statement maps the table proxy_Customers to a new table Customers in a fictitious Microsoft Access database called MyAccessDB:

```
CREATE TABLE proxy_Customers
(
    ID            INTEGER CONSTRAINT PRIMARY KEY,
    ClientName    TEXT,
    ClientAddress TEXT,
    Telephone     TEXT
)
AT 'MyAccessDB;;;Customers';
```

The string in the AT clause can also contain local or global variable names enclosed in braces ({*variable-name*}). The SQL variable name must be of type CHAR, VARCHAR, or LONG VARCHAR. For example, an AT clause that contains 'access;{@myfile};;a1' indicates that @myfile is a SQL variable and that the current contents of the @myfile variable should be substituted when the proxy table is created.

Foreign key definitions are ignored on remote tables. Foreign key definitions on local tables that refer to remote tables are also ignored. Primary key definitions are sent to the remote server if the database server supports primary keys.

SHARE BY ALL clause Use this clause only when creating global temporary tables to allow the table to be shared by all connections to the database. If the SHARE BY ALL clause is specified, either ON COMMIT PRESERVE ROWS or NOT TRANSACTIONAL must be specified.

WITH [NO] DATA clause Use this clause only when you have specified the AS clause to specify whether or not to copy data from the original table into the new table.

IF NOT EXISTS clause No changes are made if the named table already exists, and an error is not returned.

column-definition Define a column in the table. The following are part of column definitions.

- **column-name** The column name is an identifier. Two columns in the same table cannot have the same name.
- **data-type** The type of data stored in the column. Define the data type explicitly, or specify the %TYPE attribute to set the data type to the data type of a column in another table or view. When you specify the %TYPE attribute, other attributes on the object referenced in the %TYPE specification, such as default values, constraints, and whether NULLs are allowed, are not part of the definition that is inherited and must be specified separately.
- **COMPRESSED clause** Compress the column.

INLINE and PREFIX clauses The INLINE clause specifies the maximum BLOB size, in bytes, to store within the row. BLOBs smaller than or equal to the value specified by the INLINE clause are stored within the row. BLOBs that exceed the value specified by the INLINE clause are stored outside the row in table extension pages. Also, a copy of some bytes from the beginning of the BLOB may be kept in the row when a BLOB is larger than the INLINE value. Use the PREFIX clause to specify how many bytes are kept in the row. The PREFIX clause can improve the performance of requests that need the prefix bytes of a BLOB to determine if a row is accepted or rejected.

The prefix data for a compressed column is stored uncompressed, so if all the data required to satisfy a request is stored in the prefix, no decompression is necessary.

If neither INLINE nor PREFIX is specified, or if USE DEFAULT is specified, default values are applied as follows:

- For character data type columns, such as CHAR, NCHAR, and LONG VARCHAR, the default value of INLINE is 256, and the default value of PREFIX is 8.
- For binary data type columns, such as BINARY, LONG BINARY, VARBINARY, BIT, VARBIT, LONG VARBIT, BIT VARYING, and UUID, the default value of INLINE is 256, and the default value of PREFIX is 0.

Note It is strongly recommended that you use the default values unless there are specific circumstances that require a different setting. The default values have been chosen to balance performance and disk space requirements. For example, if you set INLINE to a large value, and all the BLOBs are stored inline, row processing performance may degrade. If you set PREFIX too high, you increase the amount of disk space required to store BLOBs since the prefix data is a duplicate of a portion of the BLOB.

If only one of the values is specified, the other value is automatically set to the largest amount that does not conflict with the specified value. Neither the INLINE nor PREFIX value can exceed the database page size. Also, there is a small amount of overhead reserved in a table page that cannot be used to store row data. Therefore, specifying an INLINE value approximate to the database page size can result in a slightly smaller number of bytes being stored inline.

INDEX and NO INDEX clauses When storing BLOBs (character or binary types only), specify INDEX to create BLOB indexes on inserted values that exceed the internal BLOB size

threshold (approximately eight database pages). This is the default behavior.

BLOB indexes can improve performance when random access searches within the BLOBs are required. However, for some types of BLOB values, such as images and multimedia files that will never require random-access, performance can improve if BLOB indexing is turned off. To turn off BLOB indexing for a column, specify NO INDEX.

Note A BLOB index is not the same as a table index. A table index is created to index values in one or more columns.

NULL and NOT NULL clauses If NULL is specified, NULL values are allowed in the column. This is the default behavior. This setting is controlled by the `allow_nulls_by_default` database option.

By default, columns declared as BIT are not nullable, but they can be explicitly made nullable.

If NOT NULL is specified, NULL values are not allowed.

If the column is part of a UNIQUE or PRIMARY KEY constraint, the column cannot contain NULL, even if NULL is specified.

DEFAULT clause If a DEFAULT value is specified, it is used as the value for the column in any INSERT statement that does not specify a value for the column. If no DEFAULT value is specified, it is equivalent to DEFAULT NULL.

Following is a list of possible values for DEFAULT:

- **constant-expression** Constant expressions that do not reference database objects are allowed in a DEFAULT clause, so 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.
- **global-variable** A global variable.
- **sequence-expression** You can set DEFAULT to the current value or next value from a sequence in the database.
- **string** A string value.
- **AUTOINCREMENT** When using AUTOINCREMENT, the column must be one of the integer data types, or an exact numeric type.

On inserts into the table, if a value is not specified for the AUTOINCREMENT column, a unique value larger than any other value in the column is generated. If an INSERT specifies a value for the column that is larger than the current maximum value for the column, that value is inserted and then 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 an explicit insert of a column value less than the maximum, subsequent rows without explicit assignment are still automatically incremented with a value of one greater than the previous maximum.

You can find the most recently inserted value of the column by inspecting the @@identity global variable.

AUTOINCREMENT values are maintained as signed 64-bit integers, corresponding to the data type of the `max_identity` column in the SYSTABCOL system view. When the next value to be generated exceeds the maximum value that can be stored in the column to which the AUTOINCREMENT is assigned, NULL is returned. If the column has been declared to not allow NULLs, as is true for primary key columns, a SQL error is generated.

The next value to use for a column can be reset using the `sa_reset_identity` procedure.

- o **GLOBAL AUTOINCREMENT** This default is intended for use when multiple databases are used in a MobiLink synchronization environment or SQL Remote replication.

This clause is similar to `AUTOINCREMENT`, except that the domain is partitioned. Each partition contains the same number of values. You assign each copy of the database a unique global database identification number. The database server supplies default values in a database only from the partition uniquely identified by that database's number.

The partition size can be specified in parentheses immediately following the `AUTOINCREMENT` keyword. The partition size can be any positive integer, although the partition size is generally chosen so that the supply of numbers within any one partition will rarely, if ever, be exhausted.

If the column is of type `BIGINT` or `UNSIGNED BIGINT`, the default partition size is $2^{32} = 4294967296$; for columns of all other types, the default partition size is $2^{16} = 65536$. Since these defaults may be inappropriate, especially if your column is not of type `INT` or `BIGINT`, it is best to specify the partition size explicitly.

When using this default, the value of the public option `global_database_id` in each database must be set to a unique, non-negative integer. This value uniquely identifies the database and indicates from which partition default values are to be assigned. The range of allowed values is $n p + 1$ to $p(n + 1)$, where n is the value of the public option `global_database_id` and p is the partition size. For example, if you define the partition size to be 1000 and set `global_database_id` to 3, then the range is from 3001 to 4000.

If the previous value is less than $p(n + 1)$, the next default value is one greater than the previous largest value in the column. If the column contains no values, the first default value is $n p + 1$. Default column values are not affected by values in the column outside the current partition; that is, by numbers less than $n p + 1$ or greater than $p(n + 1)$. Such values may be present if they have been replicated from another database via MobiLink or SQL Remote.

You can find the most recently inserted value of the column by inspecting the `@@identity` global variable.

`GLOBAL AUTOINCREMENT` values are maintained as signed 64-bit integers, corresponding to the data type of the `max_identity` column in the `SYSTABCOL` system view. When the supply of values within the partition has been exhausted, `NULL` is returned. If the column has been declared to not allow `NULL`s, as is true for primary key columns, a SQL error is generated. In this case, a new value of `global_database_id` should be assigned to the database to allow default values to be chosen from another partition. To detect that the supply of unused values is low and handle this condition, create an event of type `GlobalAutoincrement`.

Because the public option `global_database_id` cannot be set to a negative value, the values chosen are always positive. The maximum identification number is restricted only by the column data type and the partition size.

If the public option `global_database_id` is set to the default value of 2147483647, a `NULL` value is inserted into the column. If `NULL` values are not permitted, attempting to insert the row causes an error.

The next value to use for a column can be reset using the `sa_reset_identity` procedure.

- o **special-value** You use one of several special values in the `DEFAULT` clause (for

example, CURRENT DATE) including, but not limited to, the following special values.

- **[CURRENT] TIMESTAMP** Provides a way of indicating when each row in the table was last modified. When a column is declared with DEFAULT TIMESTAMP, a default value is provided for inserts, and the value is updated with the current date and time of day whenever the row is updated.

To provide a default value on insert, but not update the column whenever the row is updated, use DEFAULT CURRENT TIMESTAMP instead of DEFAULT TIMESTAMP.

Columns declared with DEFAULT TIMESTAMP contain unique values, so that applications can detect near-simultaneous updates to the same row. If the current TIMESTAMP value is the same as the last value, it is incremented by the value of the default_timestamp_increment option.

You can automatically truncate TIMESTAMP values based on the default_timestamp_increment option. This is useful for maintaining compatibility with other database software that records less precise timestamp values.

The global variable @@dbts returns a TIMESTAMP value representing the last value generated for a column using DEFAULT TIMESTAMP.

- **[CURRENT] UTC TIMESTAMP** Provides a way of indicating when each row in the table was last modified. When a column is declared with DEFAULT UTC TIMESTAMP, a default value is provided for inserts, and the value is updated with the current Coordinated Universal Time (UTC) whenever the row is updated.

To provide a default value on insert, but not update the column whenever the row is updated, use DEFAULT CURRENT UTC TIMESTAMP instead of DEFAULT UTC TIMESTAMP.

The behavior of this default is the same as TIMESTAMP and CURRENT TIMESTAMP except that the date and time of day is in Coordinated Universal Time (UTC).

The global variable @@dbts does not get updated when using CURRENT UTC TIMESTAMP.

- **LAST USER** LAST USER is the user ID of the user who last modified the row.

LAST USER can be used as a default value in columns with character data types.

On INSERT, this default has the same effect as CURRENT USER.

On UPDATE, if a column with a default of LAST USER is not explicitly modified, it is changed to the name of the current user.

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

IDENTITY clause IDENTITY is a Transact-SQL-compatible alternative to using DEFAULT AUTOINCREMENT. A column defined with IDENTITY is implemented as DEFAULT AUTOINCREMENT.

column-constraint and table-constraint clauses Column and table constraints help ensure the integrity of data in the database. If a statement would cause a violation of a constraint, execution of the statement does not complete, any changes made by the statement before error detection are undone, and an error is reported. There are two classes of constraints that can be created: **check constraint**, and **referential integrity (RI) constraints**. Check constraints are used to specify conditions that must be satisfied by values of columns being put into the database. RI constraints establish a relationship between data in different tables that

must be maintained in addition to specifying uniqueness requirements for data.

There are three types of RI constraints: primary key, foreign key, and unique constraint. When you create an RI constraint (primary key, foreign key or unique constraint), the database server enforces the constraint by implicitly creating an index on the columns that make up the key of the constraint. The index is created on the key for the constraint as specified. A key consists of an ordered list of columns and a sequencing of values (ASC/DESC) for each column.

Constraints can be specified on columns or tables. A column constraint refers to one column in a table, while a table constraint can refer to one or more columns in a table.

- o **PRIMARY KEY clause** A primary key uniquely defines each row in the table. Primary keys comprise one or more columns. A table cannot have more than one primary key. In a *column-constraint* clause, specifying PRIMARY KEY indicates that the column is the primary key for the table. In a *table-constraint*, you use the PRIMARY KEY clause to specify one or more columns that, when combined in the specified order, make up the primary key for the table.

The ordering of columns in a primary key need not match the respective ordinal numbers of the columns. That is, the columns in a primary key need not have the same physical order in the row. Additionally, you cannot specify duplicate column names.

When you create a primary key, an index for the key is automatically created. You can specify the sequencing of values in the index by specifying ASC (ascending) or DESC (descending) for each column. You can also specify whether to cluster the index, using the CLUSTERED keyword.

Columns included in primary keys cannot allow NULL. Each row in the table has a unique primary key value.

It is recommended that you do not use approximate data types such as FLOAT and DOUBLE for primary keys. Approximate numeric data types are subject to rounding errors after arithmetic operations.

Spatial columns cannot be included in a primary key.

- o **FOREIGN KEY clause** A foreign key restricts the values for a set of columns to match the values in a primary key or a unique constraint of another table (the primary 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 Customers table.

The foreign key column order does not need to reflect the order of columns in the table.

Duplicate column names are not allowed in the foreign key specification.

When you create a foreign key, an index for the key is automatically created. You can specify the sequencing of values in the index by specifying ASC (ascending) or DESC (descending) for each column. You can also specify whether to cluster the index, using the CLUSTERED keyword.

A global temporary table cannot have a foreign key that references a base table and a base table cannot have a foreign key that references a global temporary table.

If you attempt to add a foreign key to a column that does not exist, that column is automatically created.

- o **NOT NULL clause** Disallow NULLs in the foreign key columns. A NULL in a foreign key means that no row in the primary table corresponds to this row in the foreign table.

- **role-name clause** The role name is the name of the foreign key. The main function of the role name is to distinguish between 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, 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.
- **REFERENCES clause** A foreign key constraint can be implemented using a REFERENCES column constraint (single column only) or a FOREIGN KEY table constraint, in which case the constraint can specify one or more columns. If you specify *column-name* in a REFERENCES column constraint, it must be a column in the primary table, must be subject to a unique constraint or primary key constraint, and that constraint must consist of only that one column. If you do not specify *column-name*, the foreign key column references the single primary key column of the primary table.
- **MATCH clause** The MATCH clause determines what is considered a match when using a multi-column foreign key by allowing you to regulate what constitutes an orphaned row versus what constitutes a violation of referential integrity. The MATCH clause also allows you to specify uniqueness for the key, thereby eliminating the need to declare uniqueness separately.

The following is a list of MATCH types you can specify.

- **MATCH [UNIQUE] SIMPLE** A match occurs for a row in the foreign key table if all the column values match the corresponding column values present in a row of the primary key table. A row is orphaned in the foreign key table if at least one column value in the foreign key is NULL.

MATCH SIMPLE is the default behavior.

If the UNIQUE keyword is specified, the referencing table can have only one match for non-NULL key values.
- **MATCH [UNIQUE] FULL** A match occurs for a row in the foreign key table if none of the values are NULL and the values match the corresponding column values in a row of the primary key table. A row is orphaned if all column values in the foreign key are NULL.

If the UNIQUE keyword is specified, the referencing table can have only one match for non-NULL key values.
- **UNIQUE clause** In a *column-constraint* clause, a UNIQUE constraint specifies that the values in the column must be unique. In a *table-constraint* clause, the UNIQUE constraint identifies one or more columns that uniquely identify each row in the table. No two rows in the table can have the same values in all the named column(s). A table can have more than one UNIQUE constraint.

A UNIQUE constraint is not the same as a unique index. Columns of a unique index are allowed to be NULL, while columns in a UNIQUE constraint are not. Also, a foreign key can reference either a primary key or a UNIQUE constraint, but cannot reference a unique index since a unique index can include multiple instances of NULL.

Columns in a UNIQUE constraint can be specified in any order. Additionally, you can specify the sequencing of values in the corresponding index that is automatically created, by

specifying ASC (ascending) or DESC (descending) for each column. You cannot specify duplicate column names, however.

It is recommended that you do not use approximate data types such as FLOAT and DOUBLE for columns with unique constraints. Approximate numeric data types are subject to rounding errors after arithmetic operations.

You can also specify whether to cluster the constraint, using the CLUSTERED keyword.

- **action clause (CASCADE, SET NULL, SET DEFAULT, RESTRICT)** The referential integrity action to take when updates and deletes would affect foreign keys.
 - **CASCADE** For ON UPDATE, this action updates all foreign keys that reference the updated primary key to the new value. For ON DELETE, this action deletes all foreign key rows that reference the deleted primary key.
 - **SET NULL** Sets all foreign keys that reference the modified primary key to NULL.
 - **SET DEFAULT** Sets all foreign keys that reference the modified primary key to the default value for that column (as specified in the table definition).
 - **RESTRICT** Generates an error and prevents the modification if an attempt to alter a referenced primary key value occurs. This is the default referential integrity action.
- **CHECK clause** This 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 M or F.

If you need to create a CHECK constraint that involves a relationship between two or more columns in the table (for example, column A must be less than column B), define a table constraint instead.

No row in a table is allowed to violate a CHECK constraint. If an INSERT or UPDATE statement would cause a row to violate the constraint, the operation is not permitted and the effects of the statement are undone. The change is rejected only if a CHECK constraint condition evaluates to FALSE, and the change is allowed if a CHECK constraint condition evaluates to TRUE or UNKNOWN.

- **COMPUTE clause** The COMPUTE clause is only for use in a *column-constraint* clause. When a column is created using a COMPUTE clause, its value in any row is the value of the supplied expression. Columns created with this constraint are read-only columns for applications: the value is changed by the database server whenever the row is modified. The COMPUTE expression should not return a non-deterministic value. For example, it should not include a special value such as CURRENT_TIMESTAMP, or a non-deterministic function. If a COMPUTE expression returns a non-deterministic value, then it cannot be used to match an expression in a query.

The COMPUTE clause is ignored for remote tables.

Any UPDATE statement that attempts to change the value of a computed column fires any triggers associated with the column.

CHECK ON COMMIT clause The CHECK ON COMMIT clause overrides the wait_for_commit database option, and causes the database server to wait for a COMMIT before checking RESTRICT actions on a foreign key. The CHECK ON COMMIT clause delays foreign key checking, but does not delay other actions such as CASCADE, SET NULL, SET DEFAULT, or check constraints.

FOR OLAP WORKLOAD clause When you specify FOR OLAP WORKLOAD in the

REFERENCES clause of a foreign key definition, the database server performs certain optimizations and gathers statistics on the key to help improve performance for OLAP workloads, particularly when the `optimization_workload` option is set to OLAP.

PCTFREE clause Specifies the percentage of free space you want to reserve for each table page. The free space is used if rows increase in size when the data is updated. If there is no free space in a table page, every increase in the size of a row on that page requires the row to be split across multiple table pages, causing row fragmentation and possible performance degradation.

The value *percent-free-space* is an integer between 0 and 100. The former value specifies that no free space is to be left on each page, each page is to be fully packed. A high value causes each row to be inserted into a page by itself. If PCTFREE is not set, or is later dropped, the default PCTFREE value is applied according to the database page size (200 bytes for a 4 KB (and up) page size). The value for PCTFREE is stored in the ISYSTAB system table.

Remarks

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 specified, the table is a temporary table. Otherwise, the table is a base table.

The CREATE TABLE...LIKE syntax creates a new table based directly on the definitions of another table. You can also clone a table with additional columns, constraints, and LIKE clauses, or create a table using a SELECT statement.

Tables created by preceding the table name in a CREATE TABLE statement with a pound sign (#) are declared temporary tables, which are available only in the current connection. Temporary tables created with the pound sign (#) are identical to those created with the ON COMMIT PRESERVE ROWS clause.

Two local temporary tables within the same scope cannot have the same name. If you create a temporary table with the same name as a base table, the base table only becomes visible within the connection once the scope of the local temporary table ends. A connection cannot create a base table with the same name as an existing temporary table.

Columns allow NULLs by default. This setting can be controlled using the `allow_nulls_by_default` database option.

Privileges

You must have the CREATE TABLE system privilege to create tables owned by you. You must have the CREATE ANY TABLE or CREATE ANY OBJECT system privilege to create tables owned by others.

To create proxy tables owned by you, you must have the CREATE PROXY TABLE system privilege. You must have the CREATE ANY TABLE or CREATE ANY OBJECT system privilege to create proxy tables owned by others.

To replace an existing table, you must be the owner of the table, or have one of the following:

- CREATE ANY TABLE and DROP ANY TABLE system privileges.
- CREATE ANY OBJECT and DROP ANY OBJECT system privileges.
- ALTER ANY OBJECT or ALTER ANY TABLE system privileges.

Side effects

Automatic commit (even when creating global temporary tables).

Standards

- **ANSI/ISO SQL Standard** CREATE TABLE is a Core Feature, though some of its components supported in the software are optional SQL Language Features. A subset of these features include:
 - Temporary table support is SQL Language Feature F531.
 - Support for IDENTITY columns is SQL Feature T174, though the software uses slightly different syntax from that in the standard.
 - Foreign key constraint support includes SQL Language Features T191 "Referential action: RESTRICT", F741 "Referential MATCH types", F191 "Referential delete actions", and F701 "Referential update actions". The software does not support MATCH PARTIAL.

The software does not support SQL Language Feature T591 ("UNIQUE constraints of possibly null columns"). In the software, all columns that are part of a PRIMARY KEY or UNIQUE constraint must be declared NOT NULL.

The following components of CREATE TABLE are not in the standard:

- The { IN | ON } *dbspace-name* clause.
- The ENCRYPTED, NOT TRANSACTIONAL, and SHARE BY ALL clauses.
- The COMPRESSED, INLINE, PREFIX, and NO INDEX clauses of a column definition.
- Various implementation-defined DEFAULT values, including AUTOINCREMENT, GLOBAL AUTOINCREMENT, CURRENT DATABASE, CURRENT REMOTE USER, CURRENT UTC TIMESTAMP, and most special values. A DEFAULT clause that references a SEQUENCE generator is also not in the standard.
- The specification of MATCH UNIQUE.
- Sortedness specification (ASC or DESC) on a PRIMARY KEY or FOREIGN KEY clause.
- The ability to specify FOREIGN KEY columns in an order different from that specified in the referenced table's PRIMARY KEY clause.

Example

The following statement creates a table `file_table` with two columns: `file_name` and `file_contents`. The contents column is LONG BINARY and is compressed:

```
CREATE TABLE file_table (  
    file_name VARCHAR(255),  
    file_contents LONG BINARY COMPRESSED  
);
```

The following example creates 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
  -- are created automatically
);

```

The following example creates a table for a library database to hold information about borrowed books. The default value for date_borrowed indicates that the book is borrowed on the day the entry is made. The date_returned column is NULL until the book is returned.

```

CREATE TABLE borrowed_book (
  date_borrowed DATE NOT NULL DEFAULT CURRENT DATE,
  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 creates 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
);

```

The following example creates a table named t1 on a **fictitious** remote server, SERVER_A, and creates a proxy table named t1 that is mapped to the remote table:

```

CREATE TABLE t1
( a INT,
  b CHAR(10) )
AT 'SERVER_A.db1.joe.t1';

```

The following example creates two tables named Table1 and Table2, adds foreign keys to Table2, and inserts values into Table1. The final statement attempts to insert values into Table2. An error is returned because the values that you attempt to insert are not a simple match with Table1.

```
CREATE TABLE Table1 ( P1 INT, P2 INT, P3 INT, P4 INT, P5 INT, P6 INT, PRIMARY KEY
( P1, P2 ) );
CREATE TABLE Table2 ( F1 INT, F2 INT, F3 INT, PRIMARY KEY ( F1, F2 ) );
ALTER TABLE Table2
  ADD FOREIGN KEY fk2( F1,F2 )
  REFERENCES Table1( P1, P2 )
  MATCH SIMPLE;
INSERT INTO Table1 (P1, P2, P3, P4, P5, P6) VALUES ( 1,2,3,4,5,6 );
INSERT INTO Table2 (F1,F2) VALUES ( 3,4 );
```

The following statements show how MATCH SIMPLE and MATCH SIMPLE UNIQUE differ in how multi-column foreign keys are handled when some but not all of the columns in the key are NULL. This statement will fail because the values for the second column are not unique.

```
CREATE TABLE pt( pk INT PRIMARY KEY, str VARCHAR(10));
INSERT INTO pt VALUES(1,'one'), (2,'two');
COMMIT;

CREATE TABLE ft1( fpk INT PRIMARY KEY, FOREIGN KEY (ref) REFERENCES pt MATCH
SIMPLE);
INSERT INTO ft1 VALUES(100,1), (200,1); //This statement will insert 2 rows.

CREATE TABLE ft2( fpk INT PRIMARY KEY, FOREIGN KEY (ref) REFERENCES pt MATCH
UNIQUE SIMPLE);
INSERT INTO ft2 VALUES(100,1), (200,1);
```

The following statements show how MATCH SIMPLE and MATCH UNIQUE SIMPLE differ:


```
CREATE TABLE pt2(
    pk1 INT NOT NULL,
    pk2 INT NOT NULL,
    str VARCHAR(10),
    PRIMARY KEY (pk1,pk2));

INSERT INTO pt2 VALUES(1,10,'one-ten'), (2,20,'two-twenty');
COMMIT;

CREATE TABLE ft3(
    fpk INT PRIMARY KEY,
    ref1 INT,
    ref2 INT );

ALTER TABLE ft3 ADD FOREIGN KEY (ref1,ref2)
    REFERENCES pt2 (pk1,pk2) MATCH SIMPLE;

CREATE TABLE ft4(
    fpk INT PRIMARY KEY,
    ref1 INT,
    ref2 INT );

INSERT INTO ft3 VALUES(100,1,10);
// MATCH SIMPLE test succeeds; all column values match the corresponding values
in pt2.
INSERT INTO ft3 VALUES(200,null,null);
// MATCH SIMPLE test succeeds; at least one column in the key is null.
INSERT INTO ft3 VALUES(300,2,null);
// MATCH SIMPLE test succeeds; at least one column in the key is null.
INSERT INTO ft4 VALUES(100,1,10);
// MATCH FULL test succeeds; all column values match the corresponding values in
pt2.
INSERT INTO ft4 VALUES(200,null,null);
// MATCH FULL test succeeds; all column values in the key are null.
INSERT INTO ft4 VALUES(300,2,null);
// MATCH FULL test fails; both columns in the key must be null or match the
corresponding values in pt2.
```

Example

The second statement in the following example creates a table, myT2, and sets the data type of its column, myColumn, to the data type of the last_name column in myT1 using a %TYPE attribute. Since additional attributes such as nullability are not applied, myT2.myColumn will not have the same NOT NULL restriction that myT1.last_name has.

```
CREATE TABLE myT1
( first_name    CHAR(20),
  last_name     VARCHAR NOT NULL );
```

```
CREATE TABLE myT2
( myColumn myT1.last_name%TYPE );
```

The following example creates a table and then creates a second table based on the first table's definitions.

```
CREATE TABLE table1 ( ID INT NOT NULL DEFAULT AUTOINCREMENT, NAME LONG VARCHAR ) ;
CREATE TABLE table2 ( ADDRESS LONG VARCHAR ) ;
```

The following statement creates a table just like table1 with no data:

```
CREATE TABLE table3 LIKE table1 INCLUDING IDENTITY ;
```

The following statement creates a table like table1, but with additional columns:

```
CREATE TABLE table4 ( LIKE table1 INCLUDING IDENTITY, LIKE table2, phone LONG
  VARCHAR );
```

The following statement creates a table with any data that is in table1 and '555-5555' in the phone column for each row:

```
CREATE TABLE table5 AS ( SELECT * , '555-5555' AS phone FROM table1 ) WITH DATA ;
```

Related Information

[SQL variables](#)

The supported variables can be grouped by scope: connection, database, and global.

[Strings](#)

A string is a sequence of characters up to 2 GB in size.

[OLAP support](#)

On-Line Analytical Processing (OLAP) offers the ability to perform complex data analysis within a single SQL statement, increasing the value of the results, while improving performance by decreasing the amount of querying on the database.

[Reloading tables with AUTOINCREMENT columns](#)

You can retain the next available value for AUTOINCREMENT columns in the rebuilt database by specifying the `dbunload -I` option. This option adds calls to the `sa_reset_identity` system procedure to the generated *reload.sql* script for each table that contains an AUTOINCREMENT value, preserving the current value of `SYSTABCOL.max_identity`.

[The special IDENTITY column](#)

The value of the IDENTITY column uniquely identifies each row in a table.

[Special values](#)

Special values can be used in expressions, and as column defaults when creating tables.

[Temporary tables](#)

Temporary tables are stored in the temporary file.

[Computed columns](#)

A computed column is an expression that can refer to the values of other columns, called **dependent columns**, in the same row.

[Additional dbspaces considerations](#)

Additional database files allow you to cluster related information in separate files.

[Use of a sequence to generate unique values](#)

You can use a **sequence** to generate values that are unique across multiple tables or that are different from a set of natural numbers.

[%TYPE and %ROWTYPE attributes](#)

In addition to explicitly setting the data type for an object, you can also set the data type by specifying the %TYPE and %ROWTYPE attributes.

[Encrypting a table](#)

Create an encrypted table, or encrypt an existing table.

[Creating a table](#)

Use SQL Central to create a table in your database.

[CREATE LOCAL TEMPORARY TABLE statement](#)

Creates a local temporary table within a procedure that persists after the procedure completes and until it is either explicitly dropped, or until the connection terminates.

[@@identity global variable](#)

The @@identity variable holds the most recent value inserted by the current connection into an IDENTITY column, a DEFAULT AUTOINCREMENT column, or a DEFAULT GLOBAL AUTOINCREMENT column, or zero if the most recent insert was into a table that had no such column.

[ALTER TABLE statement](#)

Modifies a table definition or disables dependent views.

[CREATE EXISTING TABLE statement](#)

Creates a new proxy table, which represents an existing object on a remote server.

[DECLARE LOCAL TEMPORARY TABLE statement](#)

Declares a local temporary table.

[SQL data types](#)

There are many SQL data types supported by the software.

[CREATE DBSPACE statement](#)

Defines a new database space and creates the associated database file.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CREATE TEMPORARY TRACE EVENT statement

Creates a user trace event that persists until the database is stopped.

Syntax

```
CREATE [ OR REPLACE ] TEMPORARY TRACE EVENT trace-event-name  
[ WITH SEVERITY {  
    0-255  
    | ALWAYS  
    | CRITICAL  
    | ERROR  
    | WARNING  
    | INFORMATION  
    | DEBUG } ]  
[ ( field-name field-type [ , ... ] ) ]
```

Parameters

trace-event-name Specify a user trace event name. User trace event names cannot start with the prefix SYS_. System trace event names cannot be specified.

OR REPLACE clause Create a new trace event, or replace an existing trace event with the same name.

WITH SEVERITY clause If the severity level is not specified, the default severity level (DEBUG) is used. User trace events are owned by the database that the user was connected to when the trace event was created. The supported severity values are:

Level	Severity value range
ALWAYS	0
CRITICAL	1-50
ERROR	51-100
WARNING	101-150
INFORMATION	151-200
DEBUG	201-255

field-name The field that gathers information of a specific type from the user trace event. The field must be a valid identifier.

field-type You can use any data type that is supported for a column except an array type.

Remarks

A trace event is stored in memory and is dropped when the database server stops if it has not been dropped explicitly.

System privileges

You must have the MANAGE ANY TRACE SESSION system privilege.

Side effects

None

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statement creates a user trace event:

```
CREATE TEMPORARY TRACE EVENT my_event( id INTEGER, information LONG VARCHAR );
```

Related Information

[Event tracing](#)

Event tracing records information about system-defined and user-defined trace events to an event trace data (ETD) file.

[System events](#)

Each system event provides a hook on which you can program a set of actions.

[CREATE TEMPORARY TRACE EVENT SESSION statement](#)

Creates a user trace event session.

[ALTER TRACE EVENT SESSION statement](#)

Adds or removes trace events from a session, adds or removes targets from a session, or starts or stops a trace session.

[DROP TRACE EVENT statement](#)

Drops a user-defined trace event.

[DROP TRACE EVENT SESSION statement](#)

Drops a trace event session.

[NOTIFY TRACE EVENT statement](#)

Logs a user-defined trace event to a trace session.

[sp_trace_events system procedure](#)

Returns information about the trace events in the database.

[sp_trace_event_fields system procedure](#)

Returns information about the fields of the specified trace event.

[sp_trace_event_sessions system procedure](#)

Returns a list of the trace event sessions that are defined for the database.

[sp_trace_event_session_events system procedure](#)

Lists the trace events that are part of a specific trace event session.

[sp_trace_event_session_targets system procedure](#)

Lists the targets of a trace session.

[sp_trace_event_session_target_options system procedure](#)

Lists the target options for a trace event session.

[Event Trace Data \(ETD\) File Management utility \(dbmanageetd\)](#)

Generates a diagnostic log that allows the user to examine information for user-defined and system events.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

CREATE TEMPORARY TRACE EVENT SESSION statement

Creates a user trace event session.

Syntax

```
CREATE [ OR REPLACE ] TEMPORARY TRACE EVENT SESSION session-name
[ ON SERVER ]
event-definition [ ,... ]
[ target-definition ]
```

event-definition :

```
ADD TRACE EVENT event-name
```

target-definition :

```
ADD TARGET FILE
```

```
( SET target-parameter-name=target-parameter-value [ ,... ] )
```

target-parameter-name :

```
{ filename_prefix
| max_size
| num_files
| flush_on_write
| [compressed] }
```

Parameters

OR REPLACE clause Specifying CREATE OR REPLACE creates a trace event session or replaces an existing trace event session with the same name.

ON SERVER clause The trace event session records trace events from all databases on the database server. If this clause is not specified, then the trace event session only records trace events from the database on which the session is created.

session-name The name of the trace event session.

event-name The name of the trace event to add to the session. System- and user-defined trace events are supported. Call the `sp_trace_events` system procedure to obtain a list of system-defined trace events.

target-name The only supported value is FILE.

target-parameter-name The following target parameters are supported:

<i>target-parameter-name</i>	<i>target-parameter-value</i>
<code>filename_prefix</code>	An ETD file name prefix with or without a path. All ETD files have the extension <code>.etd</code> . This parameter is required.
<code>max_size</code>	The maximum size of the file in bytes. The default is 0, which means there is no limit on the file size and it grows as long as disk space is available. Once the specified size is reached, a new file is started.

<i>target-parameter-name</i>	<i>target-parameter-value</i>
num_files	Use this option to limit the number of files when max_size is set to a non-zero value. It is the number of additional files to preserve when event tracing information is split over many files. The default is 0, which means there is no limit on the number of files. When a file reaches its maximum size, the database server starts writing a new file. Each file has a file name suffix _N where N starts at 0. When num_files is not 0, older files are automatically removed. For example, if num_files is 2 and the current file number suffix is _100, then files with suffix _100, _99, and _98 are kept (the current file and 2 others). The file with suffix _97 is deleted. As each new file is written, this number suffix increases by 1.
flush_on_write	A boolean (true or false) value that controls whether disk buffers are flushed for each event that is logged. The default is false. When flushing is enabled, the performance of the database server may be reduced if many trace events are being logged.
[compressed]	A boolean (true or false) value that controls compression of the ETD file to conserve disk space. The default is false. Brackets must be used with this parameter name because it is a keyword in other contexts.

Remarks

Trace event sessions do not run until they are explicitly started with the ALTER TRACE EVENT SESSION statement. Trace event sessions can be used to capture trace events related to system behavior or for a particular user. Trace event sessions are stored in memory and are dropped when the database server stops if they have not been dropped explicitly.

System privileges

You must have the MANAGE ANY TRACE SESSION system privilege.

Side effects

None

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statement creates an event tracing session that records information about the user-defined event my_event and the system-defined event SYS_ConsoleLog_Information to a file

named *my_trace_file*:

```
CREATE TEMPORARY TRACE EVENT SESSION my_session
  ADD TRACE EVENT my_event, -- user event
  ADD TRACE EVENT SYS_ConsoleLog_Information -- system event
  ADD TARGET FILE (SET filename_prefix='my_trace_file', [compressed]=1); -- add
a target
```

Related Information

[Event tracing](#)

Event tracing records information about system-defined and user-defined trace events to an event trace data (ETD) file.

[System events](#)

Each system event provides a hook on which you can program a set of actions.

[CREATE TEMPORARY TRACE EVENT statement](#)

Creates a user trace event that persists until the database is stopped.

[ALTER TRACE EVENT SESSION statement](#)

Adds or removes trace events from a session, adds or removes targets from a session, or starts or stops a trace session.

[DROP TRACE EVENT statement](#)

Drops a user-defined trace event.

[DROP TRACE EVENT SESSION statement](#)

Drops a trace event session.

[NOTIFY TRACE EVENT statement](#)

Logs a user-defined trace event to a trace session.

[sp_trace_events system procedure](#)

Returns information about the trace events in the database.

[sp_trace_event_fields system procedure](#)

Returns information about the fields of the specified trace event.

[sp_trace_event_sessions system procedure](#)

Returns a list of the trace event sessions that are defined for the database.

[sp_trace_event_session_events system procedure](#)

Lists the trace events that are part of a specific trace event session.

[sp_trace_event_session_targets system procedure](#)

Lists the targets of a trace session.

[sp_trace_event_session_target_options system procedure](#)

Lists the target options for a trace event session.

[Event Trace Data \(ETD\) File Management utility \(dbmanagetd\)](#)

Generates a diagnostic log that allows the user to examine information for user-defined and system events.

[-sf database server option](#)

Controls whether users have access to features for databases running on the current database server.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CREATE TEXT CONFIGURATION statement

Creates a text configuration object for use with building and updating text indexes.

Syntax

```
CREATE TEXT CONFIGURATION [ owner. ] new-config-name  
FROM [ owner. ] existing-config-name
```

Parameters

FROM clause Specify the name of a text configuration object to use as the template for creating the new one. The names of the default text configuration objects are default_char and default_nchar.

When you create a text configuration object, the database options that affect how date and time values are converted to strings are copied from the default_char and default_nchar text configuration object templates.

Remarks

You create a text configuration object using another text configuration object as a template and then alter the options as needed using the ALTER TEXT CONFIGURATION statement.

To view the list of all text configuration objects in the database, and their settings, query the SYSTEXTCONFIG system view.

Privileges

You must have the CREATE TEXT CONFIGURATION system privilege to create text configurations objects owned by you. You must have the CREATE ANY TEXT CONFIGURATION or CREATE ANY OBJECT system privilege to create text configuration objects owned by others.

All text configuration objects have PUBLIC access. Any user with privilege to create a text index can also use any text configuration object.

Side effects

Automatic commit

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following CREATE TEXT CONFIGURATION statement creates a text configuration object, max_term_sixteen, using the default_char text configuration object. The subsequent ALTER TEXT CONFIGURATION statement changes the maximum term length for max_term_sixteen to 16.

```
CREATE TEXT CONFIGURATION max_term_sixteen FROM default_char;  
ALTER TEXT CONFIGURATION max_term_sixteen  
    MAXIMUM TERM LENGTH 16;
```

Related Information

[Full text search](#)

You can perform full text searching on tables.

[Example text configuration objects](#)

You can test how a text configuration object breaks a string into terms using the `sa_char_terms` and `sa_nchar_terms` system procedures.

[Database options that impact text configuration objects](#)

When a text configuration object is created, the current settings for the `date_format`, `time_format`, and `timestamp_format` database options are stored with the text configuration object.

[What to specify when creating or altering text configuration objects](#)

There are many settings to configure when creating or altering a text configuration object.

[Tutorial: Performing a full text search on a GENERIC text index](#)

Perform a full text search on a text index that uses a GENERIC term breaker.

[Tutorial: Performing a fuzzy full text search](#)

Perform a fuzzy full text search on a text index that uses an NGRAM term breaker.

[SYSTEXTCONFIG system view](#)

Each row in the SYSTEXTCONFIG system view describes one text configuration object, for use with the full text search feature. The underlying system table for this view is ISYSTEXTCONFIG.

[ALTER TEXT CONFIGURATION statement](#)

Alters a text configuration object.

[DROP TEXT CONFIGURATION statement](#)

Drops a text configuration object.

[sa_refresh_text_indexes system procedure](#)

Refreshes all text indexes defined as MANUAL REFRESH or AUTO REFRESH.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CREATE TEXT INDEX statement

Creates a text index.

Syntax

```
CREATE TEXT INDEX [ IF NOT EXISTS ] text-index-name  
ON [ owner. ] { table-name | mv-name } ( column-name, ... )  
[ IN dbspace-name ]  
[ CONFIGURATION [ owner. ] text-configuration-name ]  
[ { IMMEDIATE REFRESH  
  | MANUAL REFRESH  
  | AUTO REFRESH [ EVERY integer { MINUTES | HOURS } ] } ]
```

Parameters

IF NOT EXISTS clause When the IF NOT EXISTS clause is specified and the named text index exists, no changes are made and an error is not returned.

ON clause Specify the table and columns on which to build the text index.

IN clause Specify the dbspace in which the text index is located. If this clause is not specified, then the text index is created in the same dbspace as the table it references.

CONFIGURATION clause Specify the text configuration object to use when creating the text index. If this clause is not specified, the default_nchar text configuration object is used if any of the columns in the index are NCHAR; otherwise, the default_char text configuration object is used.

REFRESH clause Specify the refresh type for the text index. If you do not specify a REFRESH clause, IMMEDIATE REFRESH is used as the default. You can specify the following refresh types:

- **IMMEDIATE REFRESH** Refreshes the text index each time changes in the underlying table or the materialized view impact data in the text index.
- **AUTO REFRESH** Refreshes the text index automatically using an internal server event. Use the EVERY sub-clause to specify the refresh interval in minutes or hours. If you specify AUTO REFRESH without supplying interval information, the database server refreshes the text index every 60 minutes. A text index may be refreshed earlier than the interval specified by the AUTO REFRESH clause if the pending_size value, as returned by the sa_text_index_stats system procedure, exceeds 20% of the text index size at the last refresh or if the deleted_length exceeds 50% of the text index size. An internal event executes once per minute to check this condition for all AUTO REFRESH text indexes.
- **MANUAL REFRESH** The text index is refreshed manually.

Remarks

Creating more than one text index referencing a column can return unexpected results, and is not recommended.

You cannot create a text index on a regular view or a temporary table.

Once a text index is created on a materialized view, it cannot be refreshed or truncated, it can only be dropped. The text index on a materialized view is maintained by the database server whenever

the underlying materialized view is refreshed or updated.

This statement cannot be executed when there are cursors opened with the WITH HOLD clause that use either statement or transaction snapshots.

An IMMEDIATE REFRESH text index on a base table is populated at creation time and an exclusive lock is held on the table during this initial refresh. IMMEDIATE REFRESH text indexes provide full support for queries that use snapshot isolation.

An IMMEDIATE REFRESH text index on a materialized view is populated at creation time if the view is populated.

MANUAL and AUTO REFRESH text indexes must be initialized (refreshed) after creation.

Refreshes for AUTO REFRESH text indexes scan the table using isolation level 0.

Once a text index is created, you cannot change it to, or from, being defined as IMMEDIATE REFRESH. If either of these changes is required, drop and recreate the text index.

You can choose to manually refresh an AUTO REFRESH text index by using the REFRESH TEXT INDEX statement.

Privileges

To create a text index on a table, you must be the owner of the table, or have one of the following privileges:

- REFERENCES privilege on the table
- CREATE ANY INDEX system privilege
- CREATE ANY OBJECT system privilege

To create a text index on a materialized view, you must be the owner of the materialized view, or have one of the following privileges:

- CREATE ANY INDEX system privilege
- CREATE ANY OBJECT system privilege

Side effects

Automatic commit

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example creates a text index, myTxtIdx, on the Description column of the MarketingInformation table in the sample database. The MarketingTextConfig text configuration object is used, and the refresh interval is set to every 24 hours.

```
CREATE TEXT INDEX myTxtIdx ON GROUP0.MarketingInformation ( Description )  
    CONFIGURATION default_char  
    AUTO REFRESH EVERY 24 HOURS;
```

Related Information

[Snapshot isolation](#)

Snapshot isolation is designed to improve concurrency and consistency by maintaining different

versions of data.

[Text index refresh types](#)

When you create a text index, you must also choose a refresh type that is either immediate, automatic, or manual.

[Text index concepts and reference](#)

A text index stores positional information for terms in the indexed columns.

[Tutorial: Performing a full text search on a GENERIC text index](#)

Perform a full text search on a text index that uses a GENERIC term breaker.

[Viewing text index terms and settings \(SQL Central\)](#)

View text index terms and settings in SQL Central.

[Tutorial: Performing a fuzzy full text search](#)

Perform a fuzzy full text search on a text index that uses an NGRAM term breaker.

[isolation_level option](#)

Controls the locking isolation level.

[REFRESH TEXT INDEX statement](#)

Refreshes a text index.

[SYSTEXTCONFIG system view](#)

Each row in the SYSTEXTCONFIG system view describes one text configuration object, for use with the full text search feature. The underlying system table for this view is ISYSTEXTCONFIG.

[ALTER TEXT INDEX statement](#)

Alters the definition of a text index.

[DROP TEXT INDEX statement](#)

Removes a text index from the database.

[TRUNCATE TEXT INDEX statement](#)

Deletes the data in a MANUAL or an AUTO REFRESH text index.

[sa_char_terms system procedure](#)

Breaks a CHAR string into terms and returns each term as a row along with its position.

[sa_nchar_terms system procedure](#)

Breaks an NCHAR string into terms and returns each term as a row along with its position.

[sa_refresh_text_indexes system procedure](#)

Refreshes all text indexes defined as MANUAL REFRESH or AUTO REFRESH.

[sa_text_index_stats system procedure](#)

Returns statistical information about the text indexes in the database.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

CREATE TIME ZONE statement

Creates a simulated time zone that can be used by any database.

Syntax

CREATE [OR REPLACE] TIME ZONE

name time-zone-option [...]

time-zone-option :

{ **OFFSET** *offset*

| **DST OFFSET** *offset*

| **NO DST**

| [**DST**] **STARTING** *month-day-rule* **AT** { *minutes* | *hours:minutes* }

| [**DST**] **ENDING** *month-day-rule* **AT** { *minutes* | *hours:minutes* } }

offset :

'{ + | - } { *minutes* | *hours:minutes* } '

month-day-rule :

'*month* / *day-rule*'

day-rule :

number

| **LAST** *day*

| *day* >= *number*

Parameters

OR REPLACE clause Creates a new time zone or replaces an existing time zone with the same name. If the time zone already exists and OR REPLACE is not specified, then the statement fails.

offset clause Specifies the standard offset for the time zone in hours and minutes ahead of UTC. If the offset is a whole number, then it indicates minutes, so 1:00 and 60 are equivalent. The offset cannot be more than +23:59 (1439 minutes) or less than -23:59 (-1439 minutes).

DST OFFSET clause Specifies the difference from standard time that results from using daylight savings time. The default is one hour unless the NO DST clause is specified.

NO DST clause Specifies that the time zone does not observe daylight savings time. You cannot use the STARTING, ENDING, or DST OFFSET clauses with this clause.

STARTING clause Specifies the date and time when daylight savings time begins.

ENDING clause Specifies the date and time when daylight savings time ends.

- **month-day-rule** *month* is a number from 1 to 12 or an English three-letter short form (Jan, Feb, and so on).
- **day-rule** The day of the month on which to start or end daylight savings time.
 - **number** A number from 1 to 31. If specifying *number*, then *day-rule* is a date within the month. For example, specifying **Mar/12** for *month-day-rule* indicates March 12th.

- **last day** The last specified week day of the month. For example, specifying **last Fri**, indicates the last Friday of the month.
- **day** A number from 1 to 7 or an English three-letter short form (Mon, Tue, and so on).
- **day >= number** The earliest day either on or after a particular date of the month. *number* must be less than or equal to the number of days in the month. For example, **Feb/Mon>=28**.

Note *day >= number* may match a date in a different month. For example, the DST rule 'Sep / Wed >= 25' (the first Wednesday after September 25), might be in October, depending on the year.

Remarks

The DST clauses are optional. If the time zone observes daylight savings time, then the STARTING and ENDING clauses must be specified.

Privileges

You must have the MANAGE TIME ZONE system privilege to create time zones.

Side effects

Automatic commit.

Executing this statement populates the ISYSTIMEZONE system table.

Example

To add the Eastern Time zone, execute the following statement:

```
CREATE TIME ZONE EST5EDT OFFSET '-05:00'  
STARTING 'Mar/Sun>=8' AT '2:00'  
ENDING 'Nov/Sun>=1' AT '2:00';
```

The time zone is five hours behind UTC. Daylight savings time starts at 2:00 A.M. on the first Sunday on or after March 8 (the second Sunday of March), and ends at 2:00 A.M. on the first Sunday of November.

To add the Australian Eastern Time zone, execute the following statement:

```
CREATE TIME ZONE NewSouthWales OFFSET '10:00'  
STARTING 'Oct/Sun>=1' AT '2:00'  
ENDING 'Apr/Sun>=1' AT '2:00';
```

The time zone is ten hours ahead of UTC. Daylight savings time begins on the first Sunday of October at 2:00 A.M., and ends on the first Sunday of April at 2:00 A.M.

Related Information

[COMMENT statement](#)

Stores a comment for a database object in the system tables.

[ALTER TIME ZONE statement](#)

Modifies a time zone object.

[DROP TIME ZONE statement](#)

Drops a time zone from the database.

[time_zone option](#)

Specifies which time zone the database uses for time zone calculations.

[SYSTIMEZONE system view](#)

Each row in the SYSTIMEZONE system view describes one time zone. The underlying system table for this view is ISYSTIMEZONE.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)» [Alphabetical list of SQL statements](#)

CREATE TRIGGER statement

Creates a trigger on a table.

Syntax

```

CREATE [ OR REPLACE ] TRIGGER trigger-name trigger-type
{ trigger-event-list | UPDATE OF column-list }
[ ORDER integer ] ON table-name
[ REFERENCING [ OLD AS old-name ]
  [ NEW AS new-name ]
  [ REMOTE AS remote-name ] ]
[ FOR EACH { ROW | STATEMENT } ]
[ WHEN ( search-condition ) ]
trigger-body

```

column-list : *column-name*[, ...]

trigger-type :

```

BEFORE
| AFTER
| INSTEAD OF
| RESOLVE

```

trigger-event-list : *trigger-event*[, ...]

trigger-event :

```

DELETE
| INSERT
| UPDATE

```

trigger-body : a BEGIN statement that optionally includes boolean logic keywords ({ IF | ELSIF } { INSERTING | UPDATING | DELETING } THEN *some-action*)

Parameters

OR REPLACE clause Specifying OR REPLACE creates a new trigger, or replaces an existing trigger with the same name.

trigger-type Row-level triggers can be defined to execute BEFORE, AFTER, or INSTEAD OF an insert, update, or delete operation. Statement-level triggers can be defined to execute INSTEAD OF or AFTER the statement.

BEFORE UPDATE triggers fire any time an UPDATE occurs on a row, whenever the new value differs from the old value. That is, if a *column-list* is specified for a BEFORE UPDATE trigger, then the trigger fires if any of the columns in *column-list* appear in the SET clause of the UPDATE statement. If a *column-list* is specified for an AFTER UPDATE trigger, then the trigger is fired only if the value of any of the columns in *column-list* is *changed* by the UPDATE statement.

INSTEAD OF triggers are the only form of trigger that you can define on a regular view. INSTEAD OF triggers replace the triggering action with another action. When an INSTEAD OF trigger fires, the triggering action is skipped and the specified action is performed. INSTEAD OF triggers can

be defined as a row-level or a statement-level trigger. A statement-level **INSTEAD OF** trigger replaces the entire statement, including all row-level operations. If a statement-level **INSTEAD OF** trigger fires, then no row-level triggers fire as a result of that statement. However, the body of the statement-level trigger could perform other operations that, in turn, cause other row-level triggers to fire.

If you are defining an **INSTEAD OF** trigger, then you cannot use the **UPDATE OF *column-list*** clause, the **ORDER** clause, or the **WHEN** clause.

The **RESOLVE** trigger type is for use with SQL Remote; it fires before row-level **UPDATE** or **UPDATE OF *column-list*** only.

trigger-event When defining a trigger, you can combine **DELETE**, **INSERT**, and **UPDATE** events in the same definition, but triggers for **UPDATE OF** events must be defined separately. You can define any number of **DELETE**, **INSERT**, and **UPDATE** triggers on a table. You can define any number of triggers for **UPDATE OF** events on a table, but only one per column.

Triggers can be fired by the following events:

- **DELETE event** The trigger is invoked whenever one or more rows of the table are deleted.
- **INSERT event** The trigger is invoked whenever one or more rows are inserted into the table.
- **UPDATE event** The trigger is invoked whenever one or more rows of the table are updated.

The keyword **UPDATING** is also supported for this clause for compatibility with other SQL dialects. The argument for **UPDATING** is a quoted string (for example, **UPDATING('mycolumn')**), whereas the argument for **UPDATE** is an identifier (for example, **UPDATE(mycolumn)**).

- **UPDATE OF *column-list* event** The trigger is invoked whenever a row of the associated table is updated and a column in the *column-list* is modified. This type of trigger event cannot be used in a *trigger-event-list*; it must be the only trigger event defined for the trigger. This clause cannot be used in an **INSTEAD OF** trigger.

You can only specify one **UPDATE OF** trigger per column.

You can write separate triggers for each event that you need to handle or, if you have some shared actions and some actions that depend on the event, you can create a trigger for all events and use an **IF** statement to distinguish the action taking place.

ORDER clause It is good practice to specify order for triggers when defining multiple triggers on a table, even if they are not the same type. This ensures predictable results and makes easier to confirm which order they are processed in.

When defining additional triggers of the same type (insert, update, or delete) to fire at the same time (before, after, or resolve), you must specify an **ORDER** clause to tell the database server the order in which to fire the triggers. Order numbers must be unique among same-type triggers configured to fire at the same time. If you specify an order number that is not unique, then an error is returned. Order numbers do not need to be in consecutive order (for example, you could specify 1, 12, 30). The database server fires the triggers starting with the lowest number.

Typically, if you omit the **ORDER** clause, or specify 0, then the database server assigns the order of 1. However, if another same-type trigger is already set to 1, then an error is returned.

When you create additional triggers that contain *multiple* event types, if you omit the **ORDER** clause, and one or more of the event types is the same as in other triggers (for example, the

trigger-event-list for one trigger is UPDATE, INSERT, and the *trigger-event-list* for another trigger is UPDATE), the database server does not return an error. In this case, the database server processes the triggers in an implementation-specific order that may not be expected and is subject to change. Therefore, it is strongly recommended that you always specify an ORDER clause when defining more than one trigger on a table.

When adding additional triggers, you may need to modify the existing same-type triggers for the event, depending on whether the actions of the triggers interact. If they do not interact, then the new trigger must have an ORDER value unique from other existing triggers. If they do interact, you need to consider what the other triggers do, and you may need to change the order in which they fire.

The ORDER clause is not supported for INSTEAD OF triggers since there can only be one INSTEAD OF trigger of each type (insert, update, or delete) defined on a table or view.

REFERENCING clause The REFERENCING OLD and REFERENCING NEW clauses allow you to refer to the inserted, deleted, or updated rows. With this clause an UPDATE is treated as a delete followed by an insert.

An INSERT takes the REFERENCING NEW clause, which represents the inserted row. There is no REFERENCING OLD clause.

A DELETE takes the REFERENCING OLD clause, which represents the deleted row. There is no REFERENCING NEW clause.

An UPDATE takes the REFERENCING OLD clause, which represents the row before the update, and it takes the REFERENCING NEW clause, which represents the row after the update.

The meanings of REFERENCING OLD and REFERENCING NEW differ, 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 before 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 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.

FOR EACH clause 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 specify the FOR EACH STATEMENT clause if you are declaring a statement-level trigger.

WHEN clause The trigger fires only for rows where the search-condition evaluates to true. The WHEN clause can be used only with row level triggers. This clause cannot be used in an INSTEAD OF trigger.

trigger-body The trigger body contains the actions to take when the triggering action occurs, and consists of a BEGIN statement.

You can include trigger operation conditions in the BEGIN statement. Trigger operation conditions perform actions depending on the trigger event that caused the trigger to fire. For example, if the trigger is defined to fire for both updates and deletes, you can specify different actions for

the two conditions.

You can also use Boolean conditions `{ INSERTING | DELETING | UPDATING [('col-name')] }` anywhere a condition can be used in the body of the trigger. This special syntax enables you to specify an additional action to take when performing some *trigger-event*. For example, `IF INSERTING THEN SET msg = msg || 'insert'.`

Remarks

The CREATE TRIGGER statement creates a trigger associated with a table in the database, and stores the trigger in the database.

You cannot define a trigger on a materialized view. If you do, a `SQLE_INVALID_TRIGGER_MATVIEW` error is returned.

A trigger is declared as either a row-level trigger, in which case it executes before or after each row is modified, or a statement-level trigger, in which case it executes after the entire triggering statement is completed.

CREATE TRIGGER puts a table lock on the table and requires exclusive use of the table.

Privileges

You must have the CREATE ANY TRIGGER or CREATE ANY OBJECT system privilege. Additionally, you must be the owner of the table the trigger is built on or have one of the following privileges:

- ALTER privilege on the table
- ALTER ANY TABLE system privilege
- ALTER ANY OBJECT system privilege

To create a trigger on a view owned by someone else, you must have either the CREATE ANY TRIGGER or CREATE ANY OBJECT system privilege, and you must have either the ALTER ANY VIEW or ALTER ANY OBJECT system privilege.

To replace an existing trigger, you must be the owner of the table the trigger is built on, or have one of the following:

- CREATE ANY TRIGGER system privilege.
- CREATE ANY OBJECT and DROP ANY OBJECT system privileges.
- ALTER ANY OBJECT or ALTER ANY TRIGGER system privileges.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** CREATE TRIGGER is part of optional ANSI/ISO SQL Language Feature T211 "Basic trigger capability". Row triggers are optional ANSI/ISO SQL Language Feature T212, while INSTEAD OF triggers are optional ANSI/ISO SQL Language Feature T213.

Some trigger features in the software are not in the standard. These include:

- The optional OR REPLACE syntax. If an existing trigger is replaced, authorization of the creation of the new trigger instance is bypassed.
- The ORDER clause. In the ANSI/ISO SQL Standard, triggers are fired in the order they were created.
- RESOLVE triggers.

- **Transact-SQL** ROW and RESOLVE triggers are not supported by Adaptive Server Enterprise. The SQL Anywhere Transact-SQL dialect does not support Transact-SQL INSTEAD OF triggers, though these are supported by Adaptive Server Enterprise. Transact-SQL triggers are defined using different syntax.

Example

This example creates a statement-level trigger. First, create a table as shown in this CREATE TABLE statement (requires the CREATE TABLE system privilege):

```
CREATE TABLE t0
( id INTEGER NOT NULL,
  times TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP,
  remarks TEXT NULL,
  PRIMARY KEY ( id )
);
```

Next, create a statement-level trigger for this table:

```
CREATE TRIGGER myTrig AFTER INSERT ORDER 4 ON t0
REFERENCING NEW AS new_name
FOR EACH STATEMENT
BEGIN
  DECLARE @id1 INTEGER;
  DECLARE @times1 TIMESTAMP;
  DECLARE @remarks1 LONG VARCHAR;
  DECLARE @err_notfound EXCEPTION FOR SQLSTATE VALUE '02000';
  //declare a cursor for table new_name
  DECLARE new1 CURSOR FOR
    SELECT id, times, remarks FROM new_name;
  OPEN new1;
  //Open the cursor, and get the value
  LoopGetRow:
  LOOP
    FETCH NEXT new1 INTO @id1, @times1,@remarks1;
    IF SQLSTATE = @err_notfound THEN
      LEAVE LoopGetRow
    END IF;
    //print the value or for other use
    PRINT (@remarks1);
  END LOOP LoopGetRow;
  CLOSE new1
END;
```

The following example replaces the myTrig trigger created in the previous example.

```

CREATE OR REPLACE TRIGGER myTrig AFTER INSERT ORDER 4 ON t0
REFERENCING NEW AS new_name
FOR EACH STATEMENT
BEGIN
  FOR L1 AS new1 CURSOR FOR
    SELECT id, times, remarks FROM new_name
  DO
    //print the value or for other use
    PRINT (@remarks1);
  END FOR;
END;

```

The next example shows how you can use REFERENCING NEW in a BEFORE UPDATE trigger. This example ensures that postal codes in the new Employees table are in uppercase. You must have the SELECT, ALTER, and UPDATE object-level privileges on GROUP0.Employees to execute this statement:

```

CREATE TRIGGER emp_upper_postal_code
BEFORE UPDATE OF PostalCode
ON GROUP0.Employees
REFERENCING NEW AS new_emp
FOR EACH ROW
WHEN ( ISNUMERIC( new_emp.PostalCode ) = 0 )
BEGIN
  -- Ensure postal code is uppercase (employee might be
  -- in Canada where postal codes contain letters)
  SET new_emp.PostalCode = UPPER(new_emp.PostalCode)
END;

UPDATE GROUP0.Employees SET state='ON', PostalCode='n2x 4y7' WHERE EmployeeID=191;
SELECT PostalCode FROM GROUP0.Employees WHERE EmployeeID = 191;

```

The next example shows how you can use REFERENCING OLD in a BEFORE DELETE trigger. This example prevents deleting an employee from the Employees table who has not been terminated.

```

CREATE TRIGGER TR_check_delete_employee
BEFORE DELETE
ON Employees
REFERENCING OLD AS current_employee
FOR EACH ROW WHEN ( current_employee.Terminate IS NULL )
BEGIN
  RAISERROR 30001 'You cannot delete an employee who has not been fired';
END;

```

The next example shows how you can use REFERENCING NEW and REFERENCING OLD in a BEFORE UPDATE trigger. This example prevents a decrease in an employee's salary.

```
CREATE TRIGGER TR_check_salary_decrease
  BEFORE UPDATE
    ON GROUPO.Employees
  REFERENCING OLD AS before_update
    NEW AS after_update
FOR EACH ROW
BEGIN
  IF after_update.salary < before_update.salary THEN
    RAISERROR 30002 'You cannot decrease a salary';
  END IF;
END;
```

The next example shows how you can use REFERENCING NEW in a BEFORE INSERT and UPDATE trigger. The following example creates a trigger that fires before a row in the SalesOrderItems table is inserted or updated.

```
CREATE TRIGGER TR_update_date
  BEFORE INSERT, UPDATE
    ON GROUPO.SalesOrderItems
  REFERENCING NEW AS new_row
FOR EACH ROW
BEGIN
  SET new_row.ShipDate = CURRENT_TIMESTAMP;
END;
```

The following trigger displays a message on the **History** tab of the Interactive SQL **Results** pane showing which action caused the trigger to fire.

```
CREATE TRIGGER tr BEFORE INSERT, UPDATE, DELETE
ON sample_table
REFERENCING OLD AS t1old
FOR EACH ROW
BEGIN
  DECLARE msg varchar(255);
  SET msg = 'This trigger was fired by an ';
  IF INSERTING THEN
    SET msg = msg || 'insert'
  ELSEIF DELETING THEN
    set msg = msg || 'delete'
  ELSEIF UPDATING THEN
    set msg = msg || 'update'
  END IF;
  MESSAGE msg TO CLIENT
END;
```

Related Information

[INSTEAD OF triggers](#)

INSTEAD OF triggers differ from BEFORE and AFTER triggers because when an INSTEAD OF trigger

fires, the triggering action is skipped and the specified action is performed instead.

[SQL statements for implementing integrity constraints](#)

SQL statements implement integrity constraints in several ways.

[Triggers](#)

A trigger is a special form of stored procedure that is executed automatically when a statement that modifies data is executed.

[Stored procedures, triggers, batches, and user-defined functions](#)

Procedures and triggers store procedural SQL statements in a database.

[Creating a trigger on a table \(SQL Central\)](#)

Create a trigger on a table using the Create Trigger Wizard.

[BEGIN statement](#)

Specifies a compound statement.

[CREATE TRIGGER statement \[T-SQL\]](#)

Creates a new trigger in the database in a manner compatible with Adaptive Server Enterprise.

[DROP TRIGGER statement](#)

Removes a trigger from the database.

[ROLLBACK TRIGGER statement](#)

Undoes any changes made in a trigger.

[UPDATE statement](#)

Modifies rows in database tables.

[ALTER TRIGGER statement](#)

Replaces a trigger definition with a modified version. You must include the entire new trigger definition in the ALTER TRIGGER statement.

[RAISERROR statement](#)

Signals an error and sends a message to the client.

[CONFLICT function \[Miscellaneous\]](#)

Indicates if a column is a source of conflict for an UPDATE being performed against a consolidated database in a SQL Remote environment.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CREATE TRIGGER statement [T-SQL]

Creates a new trigger in the database in a manner compatible with Adaptive Server Enterprise.

Syntax

- **General use**

```
CREATE TRIGGER [owner.]trigger_name  
ON [owner.]table_name  
FOR { INSERT, UPDATE, DELETE }  
AS statement-list
```

- **Specifying additional conditions**

```
CREATE TRIGGER [owner.]trigger_name  
ON [owner.]table_name  
FOR { INSERT, UPDATE }  
AS  
[ IF UPDATE ( column-name )  
[ { AND | OR } UPDATE ( column-name ) ] ... ]  
statement-list  
[ IF UPDATE ( column-name )  
[ { AND | OR } UPDATE ( column-name ) ] ... ]  
statement-list
```

Remarks

CREATE TRIGGER acquires an exclusive table lock on the table.

The rows deleted or inserted are held in two temporary tables. In the Transact-SQL form of triggers, they can be accessed using the table names "deleted", and "inserted", as in Adaptive Server Enterprise. In the Watcom SQL CREATE TRIGGER statement, these rows are referenced using the REFERENCING clause.

Trigger names must be unique in the database.

Transact-SQL triggers are executed AFTER the triggering statement has executed.

Since the ORDER clause is not supported when creating Transact-SQL triggers, the value of *trigger_order* is set to 1. The SYSTRIGGER system table has a unique index on: *table_id*, *event*, *trigger_time*, and *trigger_order*. For a particular event (insert, update, delete), statement-level triggers are always AFTER and *trigger_order* cannot be set, so there can be only one of each type per table, assuming any other triggers do not set an order other than 1.

Privileges

You must be the owner of the table, or have ALTER privilege on the table, or have ALTER ANY TABLE system privilege. Additionally, you must have the CREATE ANY TRIGGER or CREATE ANY OBJECT system privilege

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.
- **Transact-SQL** ROW triggers are not supported by Adaptive Server Enterprise. The SQL Anywhere Transact-SQL dialect does not support Transact-SQL INSTEAD OF triggers, though these are supported by Adaptive Server Enterprise.

If an *owner* is specified for *trigger_name*, it is ignored. SQL Anywhere triggers do not have owners.

Related Information

[Transact-SQL triggers](#)

Trigger compatibility requires compatibility of trigger features and syntax.

[CREATE TRIGGER statement](#)

Creates a trigger on a table.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

CREATE USER statement

Creates a database user or group.

Syntax

```
CREATE USER user-name [ IDENTIFIED BY password ]  
[ LOGIN POLICY policy-name ]  
[ FORCE PASSWORD CHANGE { ON | OFF } ]
```

Parameters

user-name The name of the user you are creating.

IDENTIFIED BY clause The password of the user. A user without a password cannot connect to the database.

policy-name The name of the login policy to assign the user. If no login policy is specified, the DEFAULT login policy is applied.

FORCE PASSWORD CHANGE clause Controls whether the user must specify a new password when they log in. This setting overrides the password_expiry_on_next_login option setting in the user's policy.

Remarks

You do not have to specify a password for the user. A user without a password cannot connect to the database. This is useful if you are creating a group and do not want anyone to connect to the database using the group user ID. A user ID must be a valid identifier.

If you use this statement in a procedure, do not specify the password as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

Privileges

You must have the MANAGE ANY USER system privilege.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example creates a user named SQLTester with the password welcome. The SQLTester user will be required to specify a password when they log in.

```
CREATE USER SQLTester IDENTIFIED BY welcome  
FORCE PASSWORD CHANGE ON;
```

Related Information

[Login policies](#)

A **login policy** consists of a set of rules that are applied when you create a database connection for a user.

[Creating a user \(SQL Central\)](#)

Create a user by using the Create User Wizard in SQL Central.

[CREATE LOGIN POLICY statement](#)

Creates a login policy.

[ALTER LOGIN POLICY statement](#)

Alters an existing login policy.

[ALTER USER statement](#)

Alters user settings.

[COMMENT statement](#)

Stores a comment for a database object in the system tables.

[DROP LOGIN POLICY statement](#)

Drops a login policy.

[DROP USER statement](#)

Drops a user.

[GRANT statement](#)

Grant system and object-level privileges to users and roles.

[GRANT ROLE statement](#)

Grant roles to users and roles.

[min_password_length option](#)

Sets the minimum length for new passwords in the database.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

CREATE VARIABLE statement

Creates a connection- or database-scope variable.

Syntax

- **Create a connection-scope variable**

```
CREATE [ OR REPLACE ] VARIABLE identifier data-type [ { = | DEFAULT }  

initial-value ]
```

initial-value : *expression*

- **Create a database-scope variable**

```
CREATE [ OR REPLACE ] DATABASE VARIABLE [ IF NOT EXISTS ] [  

owner.]identifier data-type [ { = | DEFAULT } initial-value ]
```

initial-value : *expression*

Parameters

OR REPLACE clause Specifying the OR REPLACE clause drops the named variable if it already exists and recreates it with the new definition. OR REPLACE only replaces the value of the variable if the data type of the current and new value are the same.

Do not use this clause with the IF NOT EXISTS clause.

owner This parameter applies only to database-scope variables. Specify a valid user ID or role, or PUBLIC to set ownership of the variable. If set to a user, only that user can use the database variable. If set to a role, users who have that role are able to use the database variable. If set to PUBLIC, all users are able to use the variable.

If *owner* is not specified, it is set to the user executing the CREATE VARIABLE statement.

identifier A valid identifier for the variable.

data-type The data type for the variable. Set the data type explicitly, or use the %TYPE or %ROWTYPE attribute to set the data type to the data type of another object in the database. Use %TYPE to set it to the data type of a variable or a column in a table or view. Use %ROWTYPE to set the data type to a composite data type derived from a row in a cursor, table, or view.

%ROWTYPE and TABLE REF is not supported as data types for database-scope variables.

IF NOT EXISTS clause Specify this clause to allow the statement to complete without returning an error if a database-scope variable with the same name already exists. This parameter is only for use when creating owned database-scope variables.

Do not use this clause with the OR REPLACE clause.

{ = | DEFAULT } clause The default value for the variable. For database-scope variables, this is also the initial value after the database is restarted.

initial-value must match the data type defined by *data-type*. If you do not specify an *initial-value*, then the variable contains the NULL value until a different value is assigned, for example by using a SET statement, a SELECT ... INTO statement, or in an UPDATE statement. If *initial-value* is set by using an expression, then the expression is evaluated at creation time and the resulting

constant is stored (not the expression).

Remarks

Variables are useful for sharing values between procedures. They are also useful for creating large text or binary objects for INSERT or UPDATE statements from Embedded SQL programs.

Database-scope variables are useful for sharing values across connections and database restarts.

Use the CREATE VARIABLE syntax to create a connection-scope variable that is available in the context of the connection.

Use the CREATE DATABASE VARIABLE syntax to create a database-scope variable that can be used by other users and other connections.

Use the OR REPLACE clause as an alternative to the VAREXISTS function in SQL scripts.

If you specify a variable name for *initial-value*, then the variable must already be initialized in the database.

Privileges

Connection-scope variables: No privileges are required to create or replace a connection-scope variable.

Database-scope variables: To create or replace a self-owned database-scope variable, you must have the CREATE DATABASE VARIABLE or MANAGE ANY DATABASE VARIABLE system privilege. To create or replace a database-scope variable owned by another user or by PUBLIC, you must have the MANAGE ANY DATABASE VARIABLE system privilege.

Side effects

Connection-scope variables: There are no side effects associated with creating a connection-scope variable.

Database-scope variables: Creating and replacing a database-scope variable causes an automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

This example creates (or updates) a database-scope variable called site_name of type VARCHAR(50).

```
CREATE OR REPLACE DATABASE VARIABLE @site_name VARCHAR(50);
```

This example creates (or updates) a database-scope variable owned by PUBLIC called database_name of type CHAR(66) and sets it to the special value CURRENT DATABASE.

```
CREATE OR REPLACE DATABASE VARIABLE PUBLIC.@database_name CHAR(66) DEFAULT  
CURRENT DATABASE;
```

This example creates a connection-scope variable named first_name, of data type VARCHAR(50).

```
CREATE VARIABLE first_name VARCHAR(50);
```

This example creates a connection-scope variable named `birthday`, of data type `DATE`.

```
CREATE VARIABLE birthday DATE;
```

This example creates a connection-scope variable named `v1` as an `INT` with the initial setting of 5.

```
CREATE VARIABLE v1 INT = 5;
```

This example creates a connection-scope variable named `v1` and sets its value to 10, regardless of whether the `v1` variable already exists.

```
CREATE OR REPLACE VARIABLE v1 INT = 10;
```

This example creates a connection-scope variable, `ProductID`, and uses the `%TYPE` attribute to set its data type to the data type of the `ID` column in the `Products` table:

```
CREATE VARIABLE ProductID Products.ID%TYPE;
```

This example creates a connection-scope variable, `ItemsForSale`, and uses the `%ROWTYPE` attribute to set its data type to a composite data type comprised of the columns defined for the `Products` table. It then creates another variable, `ItemID`, and declares its type to be the data type of the `ID` column in the `ItemsForSale` variable:

```
CREATE VARIABLE ItemsForSale Products%ROWTYPE;  
CREATE VARIABLE ItemID ItemsForSale.ID%TYPE;
```

Related Information

[SQL variables](#)

The supported variables can be grouped by scope: connection, database, and global.

[Compound statements](#)

The body of a procedure or trigger is a **compound statement**.

[Special values](#)

Special values can be used in expressions, and as column defaults when creating tables.

[%TYPE and %ROWTYPE attributes](#)

In addition to explicitly setting the data type for an object, you can also set the data type by specifying the `%TYPE` and `%ROWTYPE` attributes.

[BEGIN statement](#)

Specifies a compound statement.

[SQL data types](#)

There are many SQL data types supported by the software.

[DROP VARIABLE statement](#)

Drops a SQL variable.

[SET statement](#)

Assigns a value to a SQL variable.

[VAREXISTS function \[Miscellaneous\]](#)

Returns 1 if a user-defined variable exists with the specified name. Returns 0 if no such variable exists.

[SYSDATABASEVARIABLE system view](#)

Each row in the SYSDATABASEVARIABLE system view describes one database-scope variable in the database. The underlying system table for this view is ISYSDATABASEVARIABLE.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

CREATE VIEW statement

Creates a view on the database.

Syntax

```
CREATE [ OR REPLACE ] VIEW  
[ owner. ]view-name [ ( column-name, ... ) ]  
AS query-expression  
[ WITH CHECK OPTION ]
```

Parameters

OR REPLACE clause Specifying OR REPLACE (CREATE OR REPLACE VIEW) creates a view or replaces an existing view with the same name. Existing privileges are preserved when you use the OR REPLACE clause, but INSTEAD OF triggers on the view are dropped.

If you execute a CREATE OR REPLACE VIEW statement on a view that has one or more INSTEAD OF triggers, an error is returned. Drop the trigger before altering or dropping the view.

AS clause The SELECT statement on which the view is based. The SELECT statement must not refer to local temporary tables. Also, *query-expression* can have a GROUP BY, HAVING, WINDOW, or ORDER BY clause, and can contain UNION, EXCEPT, INTERSECT, or a common table expression.

Query semantics dictate that the order of the rows returned is undefined unless the query combines an ORDER BY clause with a TOP or FIRST clause in the SELECT statement.

WITH CHECK OPTION clause 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 *query-expression*.

Remarks

Views do not physically exist in the database as tables. They are derived each time they are used. A view is derived as the result of a SELECT statement specified in a CREATE VIEW statement. In a view, specifying the user ID of the table owner is recommended to distinguish tables with the same name.

A view name can be used in place of a table name in SELECT, DELETE, UPDATE, and INSERT statements.

SELECT * can be used in the main query, a subquery, a derived table, or a subselect of the CREATE VIEW statement.

A query can specify a TOP n, FIRST, or LIMIT clause even if there is no ORDER BY clause. At most the specified number of rows are returned, but the order of the rows returned is not defined, so an ORDER BY could be specified but it is not required. When a view is used in a query, the ORDER BY in the view does not determine the order of rows in the query, even if there are no other tables in the FROM clause. That means that an ORDER BY should only be included in a view if it is needed to select which rows are included by a TOP n, FIRST, or LIMIT clause. Otherwise, the ORDER BY clause has no effect and it is ignored by the database server.

Views can be updated unless the *query-expression* defining the view contains a GROUP BY clause, a WINDOW clause, an aggregate function, or involves a set operator (UNION, INTERSECT, EXCEPT). An update to the view updates the underlying table(s).

The view's columns are given the names specified in the *column-name* list. If the column name list is not specified, view columns are given names from the SELECT list items. All items in the SELECT list must have unique names. To use names from the SELECT list items, each item must be a simple column name or have a specified alias.

The database server does permit unnamed expressions in the SELECT list of the *query-expression* referenced in the CREATE VIEW statement. Unnamed expressions in the SELECT list of the *query-expression* are assigned the name **expression**, concatenated with an integer value if more than one such expression exists. For example, the following statement would define view V with three columns (expression, expression1, and expression2), and these names would appear in the SYSCOLUMN system view for the created view V.

```
CREATE VIEW V AS
  SELECT DATEADD( DAY, 1, NOW() ), DATEADD( DAY, 2, NOW() ), DATEADD( DAY, 2,
NOW() )
  FROM SYS.DUMMY;
```

Relying on these generated names is not recommended since other views with unnamed SELECT list expressions have the identical assigned names.

Typically, a view references tables and views (and their respective attributes) that are defined in the catalog. However, a view can also reference SQL variables (with the exception of TABLE REF variables). When variables are used in the definition, when a query that references the view is executed, the value of the SQL variable is substituted for the variable. Views that reference SQL variables are called **parameterized views** since the variables act as parameters to the execution of the view. Parameterized views offer an alternative to embedding the body of an equivalent SELECT block in a query as a derived table in the query's FROM clause. Parameterized views can also be useful for queries that are embedded in stored procedures where the SQL variables referenced in the view are input parameters for the procedure.

It is not necessary for the SQL variable to exist when the CREATE VIEW statement is executed. However, if the SQL variable is not defined when a query that refers to the view is executed, an error is returned indicating that the column (variable) could not be found.

Variables of type TABLE REF (table reference variables) are not permitted in the outermost SELECT list of a view definition.

Privileges

You must have the CREATE VIEW system privilege to create views owned by you. You must have the CREATE ANY VIEW or CREATE ANY OBJECT system privilege to create views owned by others.

To replace an existing view, you must be the owner of the view, or have one of the following:

- CREATE ANY VIEW and DROP ANY VIEW system privileges.
- CREATE ANY OBJECT and DROP ANY OBJECT system privileges.
- ALTER ANY OBJECT or ALTER ANY VIEW system privileges.

If the tables referenced by the view are owned by other users, you must have the SELECT object-level privileges on those tables.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Core Feature, but some features of a view's embedded SELECT statement are optional Language Features. The ability to specify an ORDER BY clause with the top-level SELECT statement in the view definition is optional Language Feature F852. Restricting the result set of a view using SELECT TOP or LIMIT is optional Language Feature F859. Specifying WITH CHECK OPTION on a view that is not updatable, for example, the view's SELECT statement contains a derived table involving aggregation or DISTINCT, or a set operator (INTERSECT, EXCEPT, or UNION), is optional Language Feature T111.

The following extensions are also not in the ANSI/ISO SQL Standard: parameterized views, the optional OR REPLACE syntax, and the automatic generation of names for unnamed SELECT list expressions.

Example

The following example creates a view showing information for male employees only. This view has the same column names as the base table:

```
CREATE VIEW MaleEmployees
AS SELECT *
FROM GROUP0.Employees
WHERE Sex = 'M';
```

The following example creates a view showing employees and the departments they belong to:

```
CREATE VIEW EmployeesAndDepartments
AS SELECT Surname, GivenName, DepartmentName
FROM GROUP0.Employees JOIN GROUP0.Departments
ON Employees.DepartmentID = Departments.DepartmentID;
```

The following example replaces the EmployeesAndDepartments view created in the previous example. After replacing the view, the view shows the city, state, and country location for each employee:

```
CREATE OR REPLACE VIEW EmployeesAndDepartments
AS SELECT Surname, GivenName, City, State, Country
FROM GROUP0.Employees JOIN GROUP0.Departments
ON Employees.DepartmentID = Departments.DepartmentID;
```

The following example creates a parameterized view based on the variables var1 and var2, which are not attributes of the Employees or Departments tables:

```
CREATE VIEW EmployeesByState
AS SELECT Surname, GivenName, DepartmentName
FROM GROUP0.Employees JOIN GROUP0.Departments
ON Employees.DepartmentID = Departments.DepartmentID
WHERE Employees.State = var1 and Employees.Status = var2;
```

Variables can appear in the view's SELECT statement in any context where a variable is a permitted expression. For example, the following parameterized view utilizes the parameter var1 as the pattern for a LIKE predicate:

```
CREATE VIEW ProductsByDescription
AS SELECT *
FROM GROUPO.Products
WHERE Products.Description LIKE var1;
```

To use this view, define the variable var1 before executing the query that references the view. For example, the following BEGIN statement could be placed in a procedure, function, or a batch statement:

```
BEGIN
DECLARE var1 CHAR(20);
SET var1 = '%cap%';
SELECT * FROM ProductsByDescription
END
```

Related Information

[INSTEAD OF triggers](#)

INSTEAD OF triggers differ from BEFORE and AFTER triggers because when an INSTEAD OF trigger fires, the triggering action is skipped and the specified action is performed instead.

[Creating a regular view](#)

Create a view that combines data from one or more sources.

[ALTER VIEW statement](#)

Replaces a view definition with a modified version.

[SELECT statement](#)

Retrieves information from the database.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DEALLOCATE DESCRIPTOR statement [ESQL]

Frees memory associated with a SQL descriptor area.

Syntax

DEALLOCATE DESCRIPTOR *descriptor-name*

descriptor-name : *identifier*

Remarks

Frees all memory associated with a descriptor area, including the data items, indicator variables, and the structure itself.

Privileges

None.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** DEALLOCATE DESCRIPTOR is part of optional ANSI/ISO SQL Language Feature B031, "Basic dynamic SQL".

Related Information

[The SQL descriptor area \(SQLDA\)](#)

The SQLDA (SQL Descriptor Area) is an interface structure that is used for dynamic SQL statements. The structure is used to pass information regarding host variables and SELECT statement results to and from the database. The SQLDA is defined in the header file *sqlda.h*.

[ALLOCATE DESCRIPTOR statement \[ESQL\]](#)

Allocates space for a SQL descriptor area (SQLDA).

[SET DESCRIPTOR statement \[ESQL\]](#)

Describes the variables in a SQL descriptor area and to place data into the descriptor area.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DEALLOCATE statement

This statement has no effect in SQL Anywhere, and is ignored. It is provided for compatibility with Adaptive Server Enterprise and Microsoft SQL Server. Refer to your Adaptive Server Enterprise or Microsoft SQL Server documentation for more information about this statement.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

Declaration section [ESQL]

Declares host variables in an Embedded SQL program. Host variables are used to exchange data with the database.

Syntax

```
EXEC SQL BEGIN DECLARE SECTION;
```

```
C declarations
```

```
EXEC SQL END DECLARE SECTION;
```

Remarks

A declaration section is simply a section of C variable declarations surrounded by the BEGIN DECLARE SECTION and END DECLARE SECTION statements. A declaration section makes the SQL preprocessor aware of C variables that are used as host variables. Not all C declarations are valid inside a declaration section.

Privileges

None.

Standards

- **ANSI/ISO SQL Standard** Core Feature.

Example

```
EXEC SQL BEGIN DECLARE SECTION;  
char *surname, initials[5];  
int dept;  
EXEC SQL END DECLARE SECTION;
```

Related Information

[Host variables in Embedded SQL](#)

Host variables are C variables that are identified to the Embedded SQL preprocessor. Host variables can be used to send values to the database server or receive values from the database server.

[BEGIN statement](#)

Specifies a compound statement.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

DECLARE CURSOR statement [ESQL] [SP]

Declares a cursor.

Syntax

- **Embedded SQL**

```

DECLARE cursor-name
[ UNIQUE ]
[ NO SCROLL
  | DYNAMIC SCROLL
  | SCROLL
  | INSENSITIVE
  | SENSITIVE ]
CURSOR FOR
{ select-statement
  | statement-name
  | call-statement }
```

- **Stored procedures**

```

DECLARE cursor-name
[ NO SCROLL
  | DYNAMIC SCROLL
  | SCROLL
  | INSENSITIVE
  | SENSITIVE ]
CURSOR
{ FOR select-statement
  | FOR call-statement
  | USING variable-name }
```

cursor-name : *identifier*

statement-name : *identifier* | *hostvar*

variable-name : *identifier*

Parameters

UNIQUE clause When a cursor is declared UNIQUE, the query is forced to return all the columns required to uniquely identify each row. Often this means ensuring that all columns in the primary key or a uniqueness table constraint are returned. Any columns that are required but were not specified in the query are added to the result set.

A DESCRIBE done on a UNIQUE cursor sets the following additional options in the indicator variables:

- **DT_KEY_COLUMN** The column is part of the key for the row.
- **DT_HIDDEN_COLUMN** The column was added to the query because it was required to uniquely identify the rows.

NO SCROLL clause A cursor declared NO SCROLL is restricted to moving forward through the result set using FETCH NEXT and FETCH RELATIVE 0 seek operations.

As rows cannot be returned to once the cursor leaves the row, there are no sensitivity restrictions on the cursor. When a NO SCROLL cursor is requested, the database server supplies the most efficient kind of cursor, which is an asensitive cursor.

DYNAMIC SCROLL clause DYNAMIC SCROLL is the default cursor type. DYNAMIC SCROLL cursors can use all formats of the FETCH statement.

When a DYNAMIC SCROLL cursor is requested, the database server supplies an asensitive cursor. When using cursors there is always a trade-off between efficiency and consistency. Asensitive cursors provide efficient performance at the expense of consistency.

SCROLL clause A cursor declared SCROLL can use all formats of the FETCH statement. When a SCROLL cursor is requested, the database server supplies a value-sensitive cursor. With a value-sensitive cursor, a subsequent fetch of a previously fetched result row may return a warning or an error if the underlying row has been modified or deleted.

The database server must execute value-sensitive cursors in such a way that result set membership is guaranteed. DYNAMIC SCROLL cursors are more efficient and should be used unless the consistent behavior of SCROLL cursors is required.

INSENSITIVE clause 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 from concurrently executing transactions, or any other update operations from within the same transaction. INSENSITIVE cursors are not updatable.

SENSITIVE clause A cursor declared SENSITIVE is sensitive to changes to membership or values of the result set.

FOR statement-name clause Statements are named using the PREPARE statement. Cursors can be declared only for a prepared SELECT or CALL. The cursor updatability specified in the PREPARE statement is used for the cursor, unless the SQL preprocessor -m HISTORICAL option is specified.

USING variable-name clause For use within stored procedures only. The variable is a string containing a SELECT statement for the cursor. The variable must be available when the DECLARE is processed, and so must be either a parameter to the procedure, or nested inside another BEGIN...END after the variable has been assigned a value.

Remarks

Cursors are the primary means for manipulating the results of queries. The DECLARE CURSOR statement declares a cursor with the specified name for a SELECT statement or a CALL statement. In a Watcom SQL procedure, trigger, or batch, a DECLARE CURSOR statement must appear with other declarations, immediately following the BEGIN keyword. Cursor names must be unique.

If a cursor is declared inside a compound statement, it exists only for the duration of that compound statement (whether it is declared in a Watcom SQL or Transact-SQL compound statement).

When a single statement is processed, all of the DECLARE CURSOR statements must use distinct names, even if the cursors are declared in scopes that do not overlap. However, the cursor can only be used within the compound statement that declares it.

The type of cursor specified in a DECLARE CURSOR statement can dictate the execution plan

selected by the query optimizer for that statement. For example, an INSENSITIVE cursor over a SELECT statement requires the complete materialization of the result set of the SELECT statement when the cursor is opened. Moreover, the type of cursor must match the characteristics of the underlying statement. If there is a mismatch between the cursor type and the statement, then the cursor type may be changed automatically. For example, an INSENSITIVE cursor declaration conflicts with an updatable SELECT statement that specifies FOR UPDATE, since by definition INSENSITIVE cursors are read only. In this case, the cursor type is changed automatically from INSENSITIVE to an updatable, value-sensitive cursor when the cursor is opened.

If the updatability of a SELECT statement embedded in a cursor declaration is unspecified, it is determined by the setting of the `ansi_update_constraints` option.

Privileges

None.

Side effects

If the cursor type must be changed to satisfy the requirements of the underlying statement, a warning is returned when the cursor is opened.

Standards

- **ANSI/ISO SQL Standard** DECLARE CURSOR is a Core Feature. The ability to specify FOR UPDATE with SCROLL or NO SCROLL is optional ANSI/ISO SQL Language Feature F831, "Full cursor update". Using DECLARE CURSOR in an Embedded SQL program constitutes optional ANSI/ISO SQL Language Feature B031. Some cursor types are also optional ANSI/ISO SQL features. These include:
 - INSENSITIVE cursors are optional SQL language feature F791 of the ANSI/ISO SQL Standard.
 - SENSITIVE cursors are optional SQL language feature F231 of the ANSI/ISO SQL Standard.
 - Scrollable cursors are optional SQL language feature F431 of the ANSI/ISO SQL Standard.

The software supports a number of extensions to DECLARE CURSOR, as follows:

- The software supports several extensions to the FOR UPDATE clause, which the ANSI/ISO SQL Standard defines as a clause of the DECLARE CURSOR statement.
 - WITH HOLD is specified as a clause of the OPEN statement, rather than as a clause of the DECLARE CURSOR statement as defined in this ANSI/ISO SQL Standard.
 - The ANSI/ISO SQL Standard separates the notions of cursor sensitivity and scrollability, while for historical reasons the software combines the two. In the software, all cursors are forward and backward scrollable except for those declared as NO SCROLL.
 - DYNAMIC SCROLL and UNIQUE are not in the ANSI/ISO SQL Standard. DYNAMIC SCROLL has similar behavior to cursors declared as ASENSITIVE in the ANSI/ISO SQL Standard.
 - The ability to declare a cursor over a CALL statement, or with a USING clause, is not in the ANSI/ISO SQL Standard.
- **Transact-SQL** DECLARE CURSOR is supported by Adaptive Server Enterprise, but there are several behavioral differences. Adaptive Server Enterprise differentiates, as in the ANSI/ISO SQL Standard, between scrollability and sensitivity; in Adaptive Server Enterprise, cursor sensitivity options are SEMI-SENSITIVE, INSENSITIVE, or default (akin to ASENSITIVE). In Adaptive Server Enterprise, NO SCROLL cursors are the default, and all scrollable cursors are

read-only. Several features of the DECLARE CURSOR statement are not supported by Adaptive Server Enterprise. These include:

- Adaptive Server Enterprise does not support the SQL Anywhere cursor concurrency clause.

To acquire a lock on a fetched row, you must use the HOLDLOCK table hint.

- Adaptive Server Enterprise does not support DYNAMIC SCROLL or UNIQUE cursors. DYNAMIC SCROLL is similar to Adaptive Server Enterprise default cursor behavior.
- The ability to declare a cursor over a CALL statement, or with a USING clause, is not supported by Adaptive Server Enterprise.

In Adaptive Server Enterprise, Transact-SQL procedures and functions can contain multiple DECLARE CURSOR statements that use the same cursor name. In Adaptive Server Enterprise, the DEALLOCATE CURSOR statement is used to eliminate a cursor from the current scope, so that a subsequent OPEN statement can reference the correct, previously declared cursor. This feature is not supported in SQL Anywhere. In SQL Anywhere, all cursors in a given scope must have unique names. If a Transact-SQL dialect procedure contains multiple cursor declarations with the same name, the procedure parses without error. However, at execution time, if a second DECLARE CURSOR statement with the same cursor name is executed, an error occurs.

You should be aware that the TDS wire protocol for Open Client and jConnect connections does not implement true scrollable result sets. When scrolling backward through a cursor, the FETCH request may be satisfied immediately if the desired row is within a window of prefetched rows that have already been retrieved by the TDS client. If the desired row is beyond this window, however, the cursor's SELECT statement may be re-executed.

Example

The following example illustrates how to declare a scroll cursor in Embedded SQL:

```
EXEC SQL DECLARE cur_employee SCROLL CURSOR
FOR SELECT * FROM GROUP0.Employees;
```

The following example illustrates how to declare a cursor for a prepared statement in Embedded SQL:

```
EXEC SQL PREPARE employee_statement
FROM 'SELECT Surname FROM GROUP0.Employees' FOR READ ONLY;
EXEC SQL DECLARE cur_employee CURSOR
FOR employee_statement;
```

The following example illustrates the use of cursors in a stored procedure:

```
BEGIN
  DECLARE cur_employee CURSOR FOR
    SELECT Surname
    FROM GROUPO.Employees;
  DECLARE name CHAR(40);
  OPEN cur_employee;
  lp: LOOP
    FETCH NEXT cur_employee INTO name;
    IF SQLCODE <> 0 THEN LEAVE lp END IF;
    ...
  END LOOP;
  CLOSE cur_employee;
END
```

This example shows the USING clause being used as a parameter to the procedure:

```
CREATE FUNCTION GetRowCount( IN qry LONG VARCHAR )
RETURNS INT
BEGIN
  DECLARE crsr CURSOR USING qry;
  DECLARE rowcnt INT;

  SET rowcnt = 0;
  OPEN crsr;
  lp: LOOP
    FETCH crsr;
    IF SQLCODE <> 0 THEN LEAVE lp END IF;
    SET rowcnt = rowcnt + 1;
  END LOOP;
  CLOSE crsr;
  RETURN rowcnt;
END;
```

This example shows the USING clause nested inside a BEGIN...END, after *variable-name* has been assigned a value.

```
CREATE PROCEDURE get_table_name(  
  IN id_value INT, OUT tablename CHAR(128)  
)  
BEGIN  
  DECLARE qry LONG VARCHAR;  
  
  SET qry = 'SELECT table_name FROM SYS.SYSTAB ' ||  
            'WHERE table_id=' || string(id_value);  
  BEGIN  
    DECLARE crsr CURSOR USING qry;  
    OPEN crsr;  
    FETCH crsr INTO tablename;  
    CLOSE crsr;  
  END  
END;
```

The following example returns an error as the two cursor names within the statement are not unique:

```
BEGIN  
  BEGIN  
    DECLARE MyCursor DYNAMIC SCROLL CURSOR FOR SELECT 1;  
  END;  
  BEGIN  
    DECLARE MYCursor DYNAMIC SCROLL CURSOR FOR SELECT 2;  
  END;  
END;
```

The following example returns an error since the cursor has not been declared within the statement that is trying to open it.

```
BEGIN  
  BEGIN  
    DECLARE MyCursor DYNAMIC SCROLL CURSOR FOR SELECT 1;  
  END;  
  BEGIN  
    OPEN MyCursor;  
  END;  
END;
```

Related Information

[Sensitive cursors](#)

Sensitive cursors can be used for read-only or updatable cursor types.

[Insensitive cursors](#)

These cursors have insensitive membership, order, and values. No changes made after cursor open time are visible.

[Value-sensitive cursors](#)

For value-sensitive cursors, membership is insensitive, and the order and value of the result set is

sensitive.

[Asensitive cursors](#)

These cursors do not have well-defined sensitivity in their membership, order, or values. The flexibility that is allowed in the sensitivity permits asensitive cursors to be optimized for performance.

[PREPARE statement \[ESQL\]](#)

Prepares a statement to be executed later, or defines a cursor.

[OPEN statement \[ESQL\] \[SP\]](#)

Opens a previously declared cursor to access information from the database.

[EXPLAIN statement \[ESQL\]](#)

Retrieves a text specification of the optimization strategy used for a particular cursor.

[SELECT statement](#)

Retrieves information from the database.

[CALL statement](#)

Invokes a procedure.

[FOR statement](#)

Repeats the execution of a statement list once for each row in a cursor.

[The Embedded SQL preprocessor](#)

The Embedded SQL preprocessor is an executable named *sqlpp*.

[ansi_update_constraints option](#)

Controls the range of updates that are permitted.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

DECLARE LOCAL TEMPORARY TABLE statement

Declares a local temporary table.

Syntax

```
DECLARE LOCAL TEMPORARY TABLE table-name
( { column-definition [ column-constraint ... ] | table-constraint | pctfree | like-clause }, ... )
like-clause | as-clause
[ ON COMMIT { DELETE | PRESERVE } ROWS
  | NOT TRANSACTIONAL ]
```

pctfree : **PCTFREE** *percent-free-space*

percent-free-space : *integer*

like-option :
 { **INCLUDING** | **EXCLUDING** } *option* [,...]

option :
 { **IDENTITY**
 | **DEFAULTS**
 | **CONSTRAINTS**
 | **INDEXES**
 | **PRIMARY KEY**
 | **FOREIGN KEYS**
 | **COMMENTS**
 | **STORAGE**
 | **ALL** }

as-clause :
 [(*column-name*, ...)] **AS** (*select-statement*)

Parameters

LIKE clause The default for *like-option* is to exclude (EXCLUDING) all options. If *source-table* is a view, then *like-option* is ignored if it is not applicable.

Specify INCLUDE to include any of the following column attributes from the original table in the new table:

- **IDENTITY** Include default autoincrement information.
- **DEFAULTS** Include default value expressions, excluding autoincrement.
- **CONSTRAINTS** Include column and table check and unique constraints.
- **INDEXES** Include indexes, excluding primary and foreign keys.
- **PRIMARY KEY** Include the primary key.
- **FOREIGN KEY** Include foreign keys.
- **COMMENTS** Include comments on columns.
- **STORAGE** Include the compression setting.

AS clause Use this clause to clone a table based on a SELECT statement.

ON COMMIT clause By default, the rows of a temporary table are deleted on a COMMIT. You can use the ON COMMIT clause to preserve rows on a COMMIT.

NOT TRANSACTIONAL clause A table created using this clause is not affected by either COMMIT or ROLLBACK. The NOT TRANSACTIONAL clause provides performance improvements in some circumstances because operations on non-transactional temporary tables do not cause entries to be made in the rollback log. For example, NOT TRANSACTIONAL can be useful if procedures that use the temporary table are called repeatedly with no intervening COMMITs or ROLLBACKs.

Remarks

You cannot use the REFERENCES *column-constraint* or the FOREIGN KEY table-constraint on a local temporary table.

The DECLARE LOCAL TEMPORARY TABLE statement declares a temporary table.

The DECLARE LOCAL TEMPORARY TABLE...LIKE syntax declares a new table based directly on the definitions of another table. You can also clone a table with additional columns, constraints, and LIKE clauses, or create a table based on a SELECT statement.

Tables created using DECLARE LOCAL TEMPORARY TABLE do not appear in the SYSTABLE view of the system catalog.

The rows of a declared temporary table are deleted when the table is explicitly dropped or when the table goes out of scope. You can also explicitly delete rows using TRUNCATE or DELETE.

Declared local temporary tables within compound statements exist within the compound statement. Otherwise, the declared local temporary table exists until the end of the connection.

Two local temporary tables within the same scope cannot have the same name. If you create temporary table with the same name as a base table, the base table only becomes visible within the connection once the scope of the local temporary table ends. A connection cannot create a base table with the same name as an existing temporary table.

If you want a procedure to create a local temporary table that persists after the procedure completes, use the CREATE LOCAL TEMPORARY TABLE statement instead.

Privileges

None.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** DECLARE LOCAL TEMPORARY TABLE is part of optional Language Feature F531. The PCTFREE and NOT TRANSACTIONAL clauses are not in the standard. The column and constraint definitions defined by the statement may also include extension syntax that is not in the standard. In the ANSI/ISO SQL Standard, tables created via the DECLARE LOCAL TEMPORARY TABLE statement appear in the system catalog; however, this is not the case in the software.
- **Transact-SQL** DECLARE LOCAL TEMPORARY TABLE is not supported by Adaptive Server Enterprise. In SAP Adaptive Server Enterprise, one creates a temporary table using the CREATE TABLE statement with a table name that begins with the special character #.

Example

The following example illustrates how to declare a temporary table in a stored procedure:

```
BEGIN
  DECLARE LOCAL TEMPORARY TABLE TempTab ( number INT );
  ...
END
```

Example

The second statement in the following example creates a table, myT2, and sets the data type of its column, myColumn, to the data type of the last_name column in myT1 using a %TYPE attribute. Since additional attributes such as nullability are not applied, myT2.myColumn does not have the same NOT NULL restriction that myT1.last_name has.

```
CREATE TABLE myT1
( first_name  CHAR(20),
  last_name   VARCHAR NOT NULL );

CREATE TABLE myT2
( myColumn myT1.last_name%TYPE );
```

The following example declares a local temporary table and then declares a second local temporary table based on the first table's definitions.

```
DECLARE LOCAL TEMPORARY TABLE table1 ( ID INT NOT NULL DEFAULT AUTOINCREMENT,
NAME LONG VARCHAR ) ;
DECLARE LOCAL TEMPORARY TABLE table2 ( ADDRESS LONG VARCHAR ) ;
```

The following statement declares a local temporary table just like table1 with no data:

```
DECLARE LOCAL TEMPORARY TABLE table3 LIKE table1 INCLUDING IDENTITY ;
```

The following statement declares a local temporary table like table1, but with additional columns:

```
DECLARE LOCAL TEMPORARY TABLE table4 ( LIKE table1 INCLUDING IDENTITY, LIKE
table2, phone LONG VARCHAR );
```

The following statement declares a local temporary table with any data that is in table1 and '555-5555' in the phone column for each row:

```
DECLARE LOCAL TEMPORARY TABLE table5 AS ( SELECT * , '555-5555' AS phone FROM
table1 ) WITH DATA ;
```

Related Information

[Compound statements](#)

The body of a procedure or trigger is a **compound statement**.

[CREATE TABLE statement](#)

Creates a new table in the database and, optionally, creates a table on a remote server.

[CREATE LOCAL TEMPORARY TABLE statement](#)

Creates a local temporary table within a procedure that persists after the procedure completes and until it is either explicitly dropped, or until the connection terminates.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

DECLARE statement

Declares a SQL variable (connection-scope) or an exception within a compound statement (BEGIN...END).

Syntax

- **Declaring a variable**

```
DECLARE identifier [, ... ] data-type  
[ { = | DEFAULT } initial-value ]
```

initial-value : *expression*

- **Declaring an exception**

```
DECLARE exception-name EXCEPTION  
FOR SQLSTATE [ VALUE ] string
```

Parameters

identifier A valid identifier for the variable.

data-type The data type for the variable. Set the data type explicitly, or you can set it by using the %TYPE or %ROWTYPE attribute. Use %TYPE to set it to the data type of a variable or a column in a table or view. Use %ROWTYPE to set the data type to a composite data type derived from a row in a cursor, table, or view.

DEFAULT clause The default value for the variable. If you specify *initial-value*, the data type must match the type defined by *data-type*.

initial-value The default and initial value for the variable. *initial-value* must match the data type defined by *data-type*. If you do not specify an *initial-value*, the variable contains the NULL value until a different value is assigned.

Remarks

DECLARE *variable-name*: Variables used in the body of a procedure, trigger, or batch can be declared using the DECLARE statement. The variable persists for the duration of the compound statement in which it is declared.

If you specify a variable name for *initial-value*, the variable must already be initialized in the database.

The body of a Watcom SQL procedure or trigger is a compound statement, and variables must be declared with other declarations, such as a cursor declaration (DECLARE CURSOR), immediately following the BEGIN keyword. In a Transact-SQL procedure or trigger, there is no such restriction.

DECLARE *exception-name* EXCEPTION: Use this syntax to declare variables for SQL language exceptions within a compound statement (BEGIN...END). The variables can be used, for example, for comparison with the SQLSTATES obtained during execution, with the SIGNAL statement, or as part of the exception case within the exception handler.

Privileges

None.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard Declaring variables syntax** - Persistent Stored Module feature. **Declaring exceptions syntax** - The form of exception declaration supported by the software, namely the DECLARE EXCEPTION statement, is not in the standard; in the ANSI/ISO SQL Standard, exceptions are specified using a handler declaration using the keywords DECLARE HANDLER. The DECLARE...EXCEPTION syntax is not allowed in T-SQL procedures.
- **Transact-SQL** The syntax for declaring exceptions cannot be used in Transact-SQL compound statements and procedures.

Example

The following batch illustrates the use of the DECLARE statement and prints a message in the database server messages window:

```
BEGIN
  DECLARE varname CHAR(61);
  SET varname = 'Test name';
  MESSAGE varname;
END
```

This example declares the following variables:

- v1 as an INT with the initial setting of 5.
- v2 and v3 as CHAR(10), both with an initial value of abc.

```
BEGIN
  DECLARE v1 INT = 5;
  DECLARE v2, v3 CHAR(10) = 'abc';
  // ...
END
```

The following procedure declares an exception for use with the SQLSTATE comparison:

```
CREATE PROCEDURE HighSales (IN cutoff INT, OUT HighValues INT)
BEGIN
  DECLARE err_notfound EXCEPTION FOR
    SQLSTATE '02000';
  DECLARE curThisCust CURSOR FOR
    SELECT CAST( sum( SalesOrderItems.Quantity *
      Products.UnitPrice ) AS INTEGER) VALUE
    FROM Customers
      LEFT OUTER JOIN SalesOrders
      LEFT OUTER JOIN SalesOrderItems
      LEFT OUTER JOIN Products
    GROUP BY CompanyName;
  DECLARE ThisValue INT;
  SET HighValues = 0;
  OPEN curThisCust;
  CustomerLoop:
  LOOP
    FETCH NEXT curThisCust
      INTO ThisValue;
    IF SQLSTATE = err_notfound THEN
      LEAVE CustomerLoop;
    END IF;
    IF ThisValue > cutoff THEN
      SET HighValues = HighValues + ThisValue;
    END IF;
  END LOOP CustomerLoop;
  CLOSE curThisCust;
END;
```

The following compound statement declares an exception for use with SIGNAL and an exception handler:

```
BEGIN
  DECLARE err_div_by_0 EXCEPTION FOR
    SQLSTATE '22012';
  DECLARE curQuantity CURSOR FOR
    SELECT Quantity
    FROM SalesOrderItems
    WHERE ProductID = 300;
  DECLARE Quantities INT;
  DECLARE altogether INT;
  SET Quantities = 0;
  SET altogether = 0;
  OPEN curQuantity;
  LOOP
    FETCH NEXT curQuantity
      INTO Quantities;
    IF SQLSTATE = '02000' THEN
      SIGNAL err_div_by_0;
    END IF;
    SET altogether = altogether + Quantities;
  END LOOP;
EXCEPTION
  WHEN err_div_by_0 THEN
    CLOSE curQuantity;
    SELECT altogether;
    return;
  WHEN OTHERS THEN
    RESIGNAL;
END;
```

The following statement creates a procedure, DepartmentsCloseToCustomerLocation, and sets its IN parameter to the data type of the ID column in the Customers table by using a %TYPE attribute:

```

CREATE OR REPLACE PROCEDURE DepartmentsCloseToCustomerLocation( IN customer_ID
Customers.ID%TYPE )
BEGIN
    DECLARE cust_rec Customers%ROWTYPE;

    SELECT City, State, Country
    INTO cust_rec.City, cust_rec.State, cust_rec.Country
    FROM Customers
    WHERE ID = customer_ID;

    SELECT Employees.Surname, Employees.GivenName, Departments.DepartmentName
    FROM Employees JOIN Departments
    ON Departments.DepartmentHeadID = Employees.EmployeeID
    WHERE Employees.City = cust_rec.City
    AND Employees.State = cust_rec.State
    AND Employees.Country = cust_rec.Country;
END;

CALL DepartmentsCloseToCustomerLocation(158);

```

The following example uses the %ROWTYPE attribute to create a variable, @a_product, and then inserts data from the Products table into the variable:

```

CREATE OR REPLACE PROCEDURE CheckStock (
    IN @id Products.ID%TYPE
)
BEGIN
    DECLARE @a_product Products%ROWTYPE;
    SET (@a_product).ID = 200;
END;

```

Related Information

[Special values](#)

Special values can be used in expressions, and as column defaults when creating tables.

[Exception handlers](#)

You can intercept certain types of errors and handle them within a procedure or trigger, rather than pass the error back to the calling environment. This is done through the use of an **exception handler**.

[%TYPE and %ROWTYPE attributes](#)

In addition to explicitly setting the data type for an object, you can also set the data type by specifying the %TYPE and %ROWTYPE attributes.

[SQL data types](#)

There are many SQL data types supported by the software.

[DECLARE statement](#)

Declares a SQL variable (connection-scope) or an exception within a compound statement (BEGIN...END).

[DECLARE CURSOR statement \[ESQL\] \[SP\]](#)

Declares a cursor.

[BEGIN statement](#)

Specifies a compound statement.

[SQLSTATE special value](#)

SQLSTATE indicates whether the most recently executed SQL statement resulted in a success, error, or warning condition.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DELETE statement (positioned) [ESQL] [SP]

Deletes the data at the current location of a cursor.

Syntax

DELETE [**[FROM]***table*] **WHERE CURRENT OF** *cursor-name*

cursor-name : *identifier* | *hostvar*

table : [*owner.*]*table-or-view* [**[AS]** *correlation-name*]

owner : *identifier*

table-or-view : *identifier*

correlation-name : *identifier*

Remarks

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.

The table from which rows are deleted is determined as follows:

- If no FROM clause is included, the cursor must be on a single table only.
- 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.
- If a FROM clause is included, *table* must unambiguously identify an updatable table in the cursor. If a *correlation-name* is specified, the server attempts to match that correlation name with a correlation name specified in the underlying cursor. If a correlation name is not specified in the DELETE statement, and a table owner is not specified, then the server attempts to match *table-or-view* with an updatable table in the underlying cursor. *table-or-view* is first matched against any correlation names.
 - If a correlation name exists in the underlying cursor, *table-or-view* may be matched with the corresponding correlation name.
 - If a correlation name does not exist, *table-or-view* must unambiguously match a table name in the cursor.
- If a FROM clause is included, and a table owner is specified, *table* must unambiguously match an updatable table in the underlying cursor.
- The positioned DELETE statement can be used on a cursor open on a view as long as the view is updatable.

Privileges

You must be the table owner, or have DELETE privilege on the table.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** The DELETE statement (positioned) is a Core Feature. The ability to use a positioned DELETE statement from within an Embedded SQL program is part of optional ANSI/ISO SQL Language Feature B031, "Basic dynamic SQL".

The FROM keyword is mandatory in the ANSI/ISO SQL Standard, but optional in the software.

The range of cursors that can be updated may contain extensions that are not in the standard if the ansi_update_constraints option is set to Off.

Example

The following statement removes the current row in the cursor cur_employee from the database.

```
DELETE  
WHERE CURRENT OF cur_employee;
```

Related Information

[UPDATE statement](#)

Modifies rows in database tables.

[UPDATE \(positioned\) statement \[ESQL\] \[SP\]](#)

Modifies the data at the current location of a cursor.

[INSERT statement](#)

Inserts a single row or a selection of rows from elsewhere in the database into a table.

[PUT statement \[ESQL\]](#)

Inserts a row into the specified cursor.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

DELETE statement

Deletes rows from the database.

Syntax

- **General use**

```
DELETE [ row-limitation ]
[ FROM ] [ owner. ] table-or-view [ [ AS ] correlation-name ]
[ WHERE search-condition ]
[ ORDER BY expression [ ASC | DESC ], ... ]
[ OPTION( query-hint, ... ) ]
```

- **Transact-SQL**

```
DELETE [ row-limitation ]
[ FROM ] [ owner. ] table-or-view [ [ AS ] correlation-name ]
[ FROM table-expression ]
[ WHERE search-condition ]
[ ORDER BY expression [ ASC | DESC ], ... ]
[ OPTION( query-hint, ... ) ]
```

table-or-view : *identifier*

row-limitation :

FIRST

| **TOP** { **ALL** | *limit-expression* } [**START AT** *startat-expression*]

limit-expression : *simple-expression*

startat-expression : *simple-expression*

simple-expression :

integer

| *variable*

| (*simple-expression*)

| (*simple-expression* { + | - | * } *simple-expression*)

query-hint :

MATERIALIZED VIEW OPTIMIZATION *option-value*

| **FORCE OPTIMIZATION**

| **FORCE NO OPTIMIZATION**

| *option-name* = *option-value*

table-expression : a full table expression that can include joins

option-name : *identifier*

option-value :
hostvar (indicator allowed)
| *string*
| *identifier*
| *number*

Parameters

row-limitation clause The row limitation clause allows you to restrict the rows being deleted to only a subset of the rows that satisfy the WHERE clause. The TOP and START AT arguments can be simple arithmetic expressions over host variables, integer constants, or integer variables. The TOP argument must evaluate to a value greater than or equal to 0. The START AT argument must evaluate to a value greater than 0. When specifying these clauses, an ORDER BY clause is required to order the rows in a meaningful manner.

FROM clause The FROM clause indicates the table from which rows will be deleted. In the Transact-SQL syntax, the second FROM clause in the DELETE statement determines the rows to be deleted from the specified table based on joins with other tables. *table-expression* can contain arbitrarily complex table expressions, including derived tables and KEY and NATURAL joins.

The following examples illustrate how correlation names are matched when the Transact-SQL syntax is used. With this statement:

```
DELETE
FROM table_1
FROM table_1 AS alias_1, table_2 AS alias_2
WHERE ...
```

table *table_1* doesn't have a correlation name in the first FROM clause but does in the second FROM clause. In this case, *table_1* in the first clause is identified with *alias_1* in the second clause. There is only one instance of *table_1* in this statement. This is allowed as 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.

However, in the following example, there are two instances of *table_1* in the second FROM clause. The statement fails with a syntax error because it is not clear which instance of the *table_1* from the second FROM clause matches the first instance of *table_1* in the first FROM clause.

```
DELETE
FROM table_1
FROM table_1 AS alias_1, table_1 AS alias_2
WHERE ...
```

WHERE clause The DELETE statement deletes all the rows that satisfy the conditions in the WHERE clause. If no WHERE clause is specified, all rows from the named table are deleted. If a second FROM clause is present, the WHERE clause qualifies the rows of the second FROM clause's *table-expression*.

ORDER BY clause Specifies the sort order for the rows to be deleted. Normally, the order in which rows are updated does not matter. However, with the FIRST or TOP clause, the order can be significant.

You cannot use ordinal column numbers in the ORDER BY clause.

Each item in the ORDER BY list can be labeled as ASC for ascending order (the default) or DESC for descending order.

OPTION clause Use this clause to specify hints for executing the statement. The setting you specify is only applicable to the current statement and takes precedence over any public or temporary option settings, including those set by ODBC-enabled applications. The following hints are supported:

- MATERIALIZED VIEW OPTIMIZATION *option-value*
- FORCE OPTIMIZATION
- FORCE NO OPTIMIZATION
- *option-name* = *option-value*.

Use an `OPTION(isolation_level = ...)` specification in the query text to override all other means of specifying isolation level for a query.

Use an `OPTION(parameterization_level = ...)` specification in the query text to override the parameterization level for a query.

Remarks

Deleting a significant amount of data using the DELETE statement causes an update to column statistics.

To delete all of the rows of a table, consider using the more efficient TRUNCATE TABLE statement.

DELETE operations can be performed on views if the query specification defining the view is updatable. A view is updatable provided the SELECT statement defining the view has only one table in the FROM clause and does not contain a DISTINCT clause, a GROUP BY clause, a WINDOW clause, an aggregate function, or involve a set operator such as UNION or INTERSECT.

Privileges

You must be the table owner, or have SELECT and DELETE privileges on the table.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Core Feature. However, the following features are not in the standard:
 - The optional FROM keyword.
 - The *row-limitation* clause and the ORDER BY clause.
 - The OPTION clause.

The Transact-SQL syntax is a Transact-SQL vendor extension.

Example

Remove all data before 2000 from the FinancialData table.

```
DELETE
FROM GROUP0.FinancialData
WHERE Year < 2000;
```

Remove the first 10 orders from SalesOrderItems table where ship date is older than 2001-01-01 and their region is Central.

```
DELETE TOP 10
FROM GROUP0.SalesOrderItems
FROM GROUP0.SalesOrders
WHERE SalesOrderItems.ID = SalesOrders.ID
    and ShipDate < '2001-01-01' and Region ='Central'
ORDER BY ShipDate ASC;
```

Remove department 600 from the database, executing the statement at isolation level 3.

```
DELETE FROM GROUP0.Departments
WHERE DepartmentID = 600
OPTION( isolation_level = 3 );
```

Related Information

[Row limitation clauses in SELECT, UPDATE, and DELETE query blocks](#)

The FIRST, TOP, and LIMIT clauses are row limitation clauses that allow you to return, update, or delete a subset of the rows that satisfy the WHERE clause.

[Regular views](#)

A view gives a name to a particular query, and holds the definition in the database system tables.

[Locks during deletes](#)

The DELETE operation follows almost the same steps as the INSERT operation, except in the opposite order.

[TRUNCATE statement](#)

Deletes all rows from a table without deleting the table definition.

[INSERT statement](#)

Inserts a single row or a selection of rows from elsewhere in the database into a table.

[INPUT statement \[Interactive SQL\]](#)

Imports data into a database table from an external file, from the keyboard, from an ODBC data source, or from a shapefile.

[FROM clause](#)

Specifies the database tables or views involved in a DELETE, SELECT, or UPDATE statement. When used within a SELECT statement, the FROM clause can also be used in a MERGE or INSERT statement.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

DESCRIBE statement [ESQL]

Gets information about the host variables required to store data retrieved from the database, or host variables required to pass data to the database.

Syntax

```
DESCRIBE
[ USER TYPES ]
[ ALL | BIND VARIABLES FOR | INPUT | OUTPUT
| SELECT LIST FOR ]
[ LONG NAMES [ long-name-spec ] | WITH VARIABLE RESULT ]
[ FOR ] { statement-name | CURSOR cursor-name }
INTO sqlda-name
```

long-name-spec :

```
OWNER.TABLE.COLUMN
| TABLE.COLUMN
| COLUMN
```

statement-name : *identifier* | *hostvar*

cursor-name : *declared cursor*

sqlda-name : *identifier*

Parameters

USER TYPES clause A DESCRIBE statement with the USER TYPES clause returns information about domains of a column. Typically, such a DESCRIBE is done when a previous DESCRIBE returns an indicator of DT_HAS_USERTYPE_INFO.

The information returned is the same as for a DESCRIBE without the USER TYPES keywords, except that the sqlname field holds the name of the domain, instead of the name of the column.

If the DESCRIBE uses the LONG NAMES clause, the sqldata field holds this information.

ALL clause DESCRIBE ALL allows you to describe INPUT and OUTPUT with one request to the database server. This has a performance benefit. The OUTPUT information is filled in the SQLDA first, followed by the INPUT 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.

BIND VARIABLES FOR clause Equivalent to the INPUT clause.

SELECT LIST FOR clause Equivalent to the OUTPUT clause.

INPUT clause 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 fills in the name fields in the SQLDA with the bind variable names. DESCRIBE INPUT also puts the number of bind variables in the sqlda 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.

OUTPUT clause The DESCRIBE OUTPUT statement fills in the data type and length for each SELECT list item in the SQLDA. 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 is derived from the SELECT list item: if the item is a simple column name, it is used; otherwise, a substring of the expression is used. DESCRIBE will also put the number of SELECT list items in the sqld field of the SQLDA.

If 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.

If you describe a CALL statement, 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 also puts the number of INOUT or OUT parameters in the sqld field of the SQLDA.

If you describe a CALL statement with a 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.

LONG NAMES clause 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. If there is an indicator variable, truncation is indicated in the usual fashion.

The default specification for the long names is TABLE.COLUMN.

WITH VARIABLE RESULT clause This clause is used to describe procedures that can have more than one result set, with different numbers or types of columns.

If WITH VARIABLE RESULT is used, the database server sets the SQLCOUNT value after the DESCRIBE statement 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 re-describing is required.

Remarks

The DESCRIBE statement sets up the named SQLDA to describe either the OUTPUT (equivalently SELECT LIST FOR) or the INPUT (equivalently BIND VARIABLES FOR) 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 you specify a statement name, the statement must have been previously prepared using the

PREPARE statement with the same statement name and the SQLDA must have been previously allocated.

If you specify a cursor name, the cursor must have been previously declared and opened. The default action is to describe the OUTPUT. Only SELECT statements and CALL statements have OUTPUT. A DESCRIBE OUTPUT on any other statement, or on a cursor that is not a dynamic cursor, indicates no output by setting the sqld field of the SQLDA to zero.

In Embedded SQL, NCHAR, NVARCHAR and LONG NVARCHAR are described as DT_FIXCHAR, DT_VARCHAR, and DT_LONGVARCHAR, respectively, by default. If the db_change_nchar_charset function has been called, these data types are described as DT_NFIXCHAR, DT_NVARCHAR and DT_LONGNVARCHAR, respectively.

Privileges

None.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** The DESCRIBE OUTPUT statement is optional ANSI/ISO SQL Language Feature B031, "Basic dynamic SQL". The DESCRIBE INPUT statement is optional ANSI/ISO SQL Language Feature B032, "Extended dynamic SQL". Many of the other clauses of the DESCRIBE statement are not in the standard. These include:
 - The USER TYPES, ALL, BIND VARIABLES FOR, LONG NAMES, and WITH VARIABLE RESULT clauses.
 - DESCRIBE uses the INTO clause to identify the sqlda; in the ANSI/ISO SQL Standard, the USING keyword is used instead.
 - In the ANSI/ISO SQL Standard, the CURSOR clause ends with the keyword STRUCTURE. STRUCTURE is not supported by the software.

Example

The following example shows how to use the DESCRIBE statement:

```
sqlda = alloc_sqlda( 3 );
EXEC SQL DESCRIBE OUTPUT
  FOR employee_statement
  INTO sqlda;
if( sqlda->sqld > sqlda->sqln ) {
  actual_size = sqlda->sqld;
  free_sqlda( sqlda );
  sqlda = alloc_sqlda( actual_size );
  EXEC SQL DESCRIBE OUTPUT
    FOR employee_statement
    INTO sqlda;
}
```

Related Information

[The SQL descriptor area \(SQLDA\)](#)

The SQLDA (SQL Descriptor Area) is an interface structure that is used for dynamic SQL statements. The structure is used to pass information regarding host variables and SELECT statement results to and from the database. The SQLDA is defined in the header file *sqlda.h*.

[ALLOCATE DESCRIPTOR statement \[ESQL\]](#)

Allocates space for a SQL descriptor area (SQLDA).

[DECLARE CURSOR statement \[ESQL\] \[SP\]](#)

Declares a cursor.

[OPEN statement \[ESQL\] \[SP\]](#)

Opens a previously declared cursor to access information from the database.

[PREPARE statement \[ESQL\]](#)

Prepares a statement to be executed later, or defines a cursor.

[db_change_nchar_charset function](#)

Change the application's NCHAR character set for this connection.

[LONG NVARCHAR data type](#)

The LONG NVARCHAR data type stores Unicode character data of arbitrary length.

[NCHAR data type](#)

The NCHAR data type stores Unicode character data, up to 32767 characters.

[NVARCHAR data type](#)

The NVARCHAR data type stores Unicode character data, up to 32767 characters.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

DESCRIBE statement [Interactive SQL]

Returns information about a given database object.

Syntax

- **Describe database objects**

```
DESCRIBE [ [ INDEX FOR ] TABLE | PROCEDURE ] [ owner.]object-name
```

```
object-name :
table
| view
| materialized view
| procedure
| function
```

- **Describe the current connection**

```
DESCRIBE CONNECTION
```

Parameters

INDEX FOR clause Indicates that you want to see the indexes for the specified *object-name*.

TABLE clause Indicates that *object-name* to be described is a table or a view.

PROCEDURE clause Indicates that *object-name* is a procedure or a function.

Remarks

Use DESCRIBE TABLE to list all the columns in the specified table or view. The DESCRIBE TABLE statement returns one row per table column, containing:

- **Column** The name of the column.
- **Type** The type of data in the column.
- **Nullable** Whether nulls are allowed (1=yes, 0=no).
- **Primary Key** Whether the column is in the primary key (1=yes, 0=no).

Use DESCRIBE INDEX FOR TABLE to list all the indexes for the specified table. The DESCRIBE TABLE statement returns one row per index, containing:

- **Index Name** The name of the index.
- **Columns** The columns in the index.
- **Unique** Whether the index is unique (1=yes, 0=no).
- **Type** The type of index. Possible values are: Clustered, Statistic, Hashed, and Other.

Use DESCRIBE PROCEDURE to list all the parameters used by the specified procedure or function. The DESCRIBE PROCEDURE statement returns one row for each parameter, containing:

- **Parameter** The name of the parameter.
- **Type** The data type of the parameter.
- **In/Out** Information about what is passed to, or returned from, the parameter. Possible values

are:

- **In** The parameter is passed to the procedure, but is not modified.
- **Out** The procedure ignores the parameter's initial value and sets its value when the procedure returns.
- **In/Out** The parameter is passed to the procedure and the procedure sets the parameter's value when the procedure returns.
- **Result** The parameter returns a result set.
- **Return** The parameter returns a declared return value.

When using DESCRIBE PROCEDURE, the returned parameter list is retrieved from the SYSPROCPARM system view.

If you do not specify either TABLE or PROCEDURE (for example, DESCRIBE *object-name*), Interactive SQL assumes the object is a table. However, if no such table exists, Interactive SQL attempts to describe the object as either a procedure or a function.

Use the DESCRIBE statement to list information about the database or database server that Interactive SQL is connected to. The following properties are returned:

- **Database Product** The name and version number of the database product Interactive SQL is connected to (for example, SQL Anywhere 17.0.0.1691).
- **Host Name** The network name of the computer the database server is running on.
- **Host TCP/IP Address** The IP address of the computer the database server is running on.
- **Host Operating System** The name and version number of the operating system used by the computer the database server is running on.
- **Server Name** The name of the database server.
- **Server TCP/IP Port** The port number used by the database server for the current connection.
- **Database Name** The name of the database that Interactive SQL is connected to.
- **Database Character Set** The character set used for CHAR columns in the database.
- **Connection String** The connection string that was used to connect to the database or database server. Three asterisks replace passwords.

Properties that do not apply to the current connection are omitted. For example, if you connect to a database server using shared memory, then the TCP/IP port is omitted.

Privileges

None

Side effects

None

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

Describe the columns in the Departments table:

DESCRIBE TABLE GROUP0.Departments;

Here is an example of the result set for this statement:

Column	Type	Nullable	Primary key
DepartmentID	integer	0	1
DepartmentName	char(40)	0	0
DepartmentHeadID	integer	1	0

List the indexes for the Customers table:

DESCRIBE INDEX FOR TABLE GROUP0.Customers;

Here is an example of the results for this statement:

Index Name	Columns	Unique	Type
IX_customer_name	Surname,GivenName	0	Clustered

Related Information

[Interactive SQL](#)

Interactive SQL is a tool that lets you execute SQL statements, run SQL script files, as well as view and compare plans.

[ALTER PROCEDURE statement](#)

Modifies a procedure.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DETACH TRACING statement (deprecated)

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database. The DETACH TRACING session ends a diagnostic tracing session.

Syntax

DETACH TRACING { WITH | WITHOUT } SAVE

Parameters

WITH SAVE clause Specify WITH SAVE to save unsaved diagnostic data in the diagnostic tables.

WITHOUT SAVE clause Specify WITHOUT SAVE if you do not want to save unsaved tracing data.

Remarks

Execute this statement from the database being profiled to stop sending diagnostic information to the diagnostic tables. If you specify the WITHOUT SAVE clause, then you can still save the data later, assuming that the tracing database is still running and another tracing session has not been started, by using the `sa_save_trace_data` system procedure.

To see the current tracing levels set for a database, look in the `sa_diagnostic_tracing_level` table.

Note Tracing information is *not* unloaded as part of a database unload or reload operation. To transfer tracing information from one database to another, you must do so manually by copying the contents of the `sa_diagnostic_*` tables; however, this is not recommended.

Privileges

You must have the `MANAGE PROFILING` system privilege.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Related Information

[Diagnostic tracing \(deprecated\)](#)

The Diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database.

[Troubleshoot and fine tune applications with the SQL Anywhere Profiler](#)

Use the Profiler to quickly view SQL statements that are currently running in your database, compare run times of procedures, and analyze how the objects in your database interact with each other.

[ATTACH TRACING statement \(deprecated\)](#)

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database. The ATTACH TRACING statement starts a diagnostic tracing session (starts sending

diagnostic information to the diagnostic tables).

[REFRESH TRACING LEVEL statement \(deprecated\)](#)

The diagnostic tracing feature is deprecated. Use the SQL Anywhere Profiler to diagnose issues in your database. The REFRESH TRACING LEVEL statement reloads the tracing levels from the sa_diagnostic_tracing_level table while a tracing session is in progress.

[sa_diagnostic_tracing_level table \(deprecated\)](#)

The sa_diagnostic_tracing_level table is owned by the dbo user, and each row in this table is a condition that determines what kind of diagnostic information to send to the tracing database.

[sa_save_trace_data system procedure](#)

Saves tracing data to base tables.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DISCONNECT statement [ESQL] [Interactive SQL]

Drops a connection to a database.

Syntax

DISCONNECT [*connection-name* | **CURRENT** | **ALL**]

connection-name :
identifier
| *string*
| *hostvar*

Remarks

The DISCONNECT statement drops a connection with the database server and releases all resources used by it. If the connection to be dropped was named on the CONNECT statement, the name can be specified. Specifying ALL will drop all the application's connections to all database environments. CURRENT is the default, and drops the current connection.

Before closing the database connection, Interactive SQL automatically executes a COMMIT statement if the commit_on_exit option is set to On. If this option is set to Off, Interactive SQL performs an implicit ROLLBACK. By default, the commit_on_exit option is set to On.

This statement is not supported in procedures, triggers, events, or batches.

Privileges

None.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** DISCONNECT comprises optional ANSI/ISO SQL Language Feature F771. The ability to specify DISCONNECT without a parameter, and the commit_on_exit option, are not in the standard.

Example

The following statement shows how to use DISCONNECT in Embedded SQL:

```
EXEC SQL DISCONNECT :conn_name
```

The following statement shows how to use DISCONNECT from Interactive SQL to disconnect all connections:

```
DISCONNECT ALL;
```

Related Information

[Interactive SQL](#)

Interactive SQL is a tool that lets you execute SQL statements, run SQL script files, as well as view and compare plans.

[DROP CONNECTION statement](#)

Drops a user's connection to the database.

[CONNECT statement \[ESQL\] \[Interactive SQL\]](#)

Establishes a connection to a database.

[SET CONNECTION statement \[Interactive SQL\] \[ESQL\]](#)

Changes the active database connection.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP CERTIFICATE statement

Drops a certificate from the database.

Syntax

DROP CERTIFICATE [**IF EXISTS**] *certificate-name*

Remarks

DROP CERTIFICATE deletes a certificate from the ISYSCERTIFICATE system table.

Use the IF EXISTS clause if you do not want an error returned when the DROP CERTIFICATE statement attempts to remove a certificate that does not exist.

Privileges

You must have the MANAGE CERTIFICATES system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

```
DROP CERTIFICATE mycert;
```

Related Information

[CREATE CERTIFICATE statement](#)

Adds or replaces a certificate in the database from the given file or string. To create a certificate, use the Certificate Creation utility (createcert).

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP CONNECTION statement

Drops a user's connection to the database.

Syntax

DROP CONNECTION *connection-id*

Remarks

The DROP CONNECTION statement disconnects a user from the database by dropping the connection to the database.

The *connection-id* parameter is an integer constant. You can obtain the *connection-id* using the `sa_conn_info` system procedure.

This statement is not supported in procedures, triggers, events, or batches.

Privileges

You must have the DROP CONNECTION system privilege.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following procedure drops a connection identified by its connection number. When executing the DROP CONNECTION statement from within a procedure, you should do so using the EXECUTE IMMEDIATE statement, as shown in this example:

```
CREATE PROCEDURE drop_connection_by_id( IN conn_number INTEGER )  
BEGIN  
    EXECUTE IMMEDIATE 'DROP CONNECTION ' || conn_number;  
END;
```

The following statement drops the connection with ID number 4.

```
DROP CONNECTION 4;
```

Related Information

[Exception handlers](#)

You can intercept certain types of errors and handle them within a procedure or trigger, rather than pass the error back to the calling environment. This is done through the use of an **exception handler**.

[CONNECT statement \[ESQL\] \[Interactive SQL\]](#)

Establishes a connection to a database.

[sa_conn_info system procedure](#)

Reports connection property information.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP DATABASE statement

Deletes all database files associated with a database.

Syntax

DROP DATABASE *database-name* [**KEY** *key*]

Remarks

The DROP DATABASE statement physically deletes all associated database files from disk. If the database file does not exist, or is not in a suitable condition for the database to be started, an error is generated.

DROP DATABASE cannot be used in stored procedures, triggers, events, or batches.

The database to be dropped must not be running when the DROP DATABASE statement is used. You cannot be connected to the database you are dropping. You must be connected to a different database. For example, connect to the utility database.

You must specify a key to drop a strongly encrypted database. The key can be either a string (constant literal) or a variable reference.

Privileges

Your ability to execute this statement depends on the setting for the -gu database option, and whether you have the SERVER OPERATOR system privilege.

Side effects

In addition to deleting the database files from disk, any associated transaction log file or transaction log mirror file is deleted.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

Execute the following statement to drop the database *temp.db*:

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

Execute the following statement to drop a database based on the constant literal, mykey:

```
DROP DATABASE 'temp.db' KEY 'mykey';
```

Execute the following statement to drop a database based on the variable reference, mykeyvar:

```
DROP DATABASE 'temp.db' KEY mykeyvar;
```

Related Information

[Connecting to the utility database \(Connect window\)](#)

Connect to the utility database to execute database file administration statements, to query

connection and server properties, and to connect to a running database server when it is not possible to connect to the database.

[Erase utility \(dberase\)](#)

Erases database files and associated transaction logs, or individual transaction log files.

[CREATE DATABASE statement](#)

Creates a database.

[DatabaseKey \(DBKEY\) connection parameter](#)

Starts an encrypted database with a connect request.

[-gu database server option](#)

Sets the privilege required for executing database file administration statements such as for creating or dropping databases.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP DATATYPE statement

Removes a data type from the database.

Syntax

DROP DATATYPE *datatype-name*

Remarks

It is recommended that you use DROP DOMAIN rather than DROP DATATYPE, as DROP DOMAIN is the syntax used in the ANSI/ISO SQL Standard. You cannot drop system-defined data types (such as MONEY or UNIQUEIDENTIFIERSTR) from a database.

Privileges

You must be the owner of the data type, or have the DROP DATATYPE or DROP ANY OBJECT system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Domain support is optional ANSI/ISO SQL Language Feature F251. The DROP DATATYPE statement is not in the standard.

Example

The following example creates and then drops a datatype called PhoneNum:

```
CREATE DATATYPE PhoneNum CHAR(12) NULL;  
DROP DATATYPE PhoneNum;
```

Related Information

[DROP DOMAIN statement](#)

Removes a domain (data type) from the database.

[CREATE DOMAIN statement](#)

Creates a domain in a database.

[ALTER DOMAIN statement](#)

Renames a user-defined domain or data type.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP DBSPACE statement

Removes a dbspace from the database.

Syntax

DROP DBSPACE *dbspace-name*

Remarks

You must drop all tables in the dbspace before dropping the dbspace. You cannot use the DROP DBSPACE statement to drop the predefined dbspaces SYSTEM, TEMPORARY, TEMP, TRANSLOG, or TRANSLOGMIRROR.

DROP DBSPACE is prevented whenever the statement affects an object that is currently being used by another connection.

You must be the only connection to the database to execute this statement.

Privileges

You must have the MANAGE ANY DBSPACE system privilege.

Side effects

Automatic commit, and causes an implicit checkpoint.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

This example drops a fictitious dbspace, MyDBSpace, from the database.

```
DROP DBSPACE MyDBSpace;
```

Related Information

[Predefined dbspaces](#)

The database server uses predefined dbspaces for its databases.

[Dropping a dbspace \(SQL Central\)](#)

Drop a dbspace in SQL Central.

[Dropping a dbspace \(SQL\)](#)

Drop a dbspace using the DROP DBSPACE statement in SQL.

[CREATE DBSPACE statement](#)

Defines a new database space and creates the associated database file.

[ALTER DBSPACE statement](#)

Preallocates space for a dbspace or for the transaction log, or updates the catalog when a dbspace file is renamed or moved.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP DOMAIN statement

Removes a domain (data type) from the database.

Syntax

DROP DOMAIN *domain-name*

Remarks

DROP DOMAIN is prevented if the data type is used in a table column, or in a procedure or function argument. You must change data types on all columns defined using the domain 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 SQL Standard. You cannot drop system-defined data types (such as MONEY or UNIQUEIDENTIFIERSTR) from a database.

Privileges

You must be the owner of the domain, or have the DROP DATATYPE or DROP ANY OBJECT system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Domain support is optional SQL Language Feature F251.

Example

The following example creates and then drops the domain CustPhoneNumber.

```
CREATE DOMAIN CustPhoneNumber CHAR(12) NULL;  
DROP DOMAIN CustPhoneNumber;
```

Related Information

[CREATE DOMAIN statement](#)

Creates a domain in a database.

[ALTER DOMAIN statement](#)

Renames a user-defined domain or data type.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP EVENT statement

Drops an event from the database.

Syntax

DROP EVENT [IF EXISTS] *event-name*

Remarks

Use the IF EXISTS clause if you do not want an error returned when the DROP EVENT statement attempts to remove an event that does not exist.

Events are not owned so do not specify an owner (an owner specification is ignored).

Privileges

You must have either the MANAGE ANY EVENT or DROP ANY OBJECT system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

This example drops a fictitious example, MyEvent, from the database.

```
DROP EVENT MyEvent;
```

Related Information

[CREATE EVENT statement](#)

Defines an event and its associated handler for automating predefined actions, and to define scheduled actions.

[ALTER EVENT statement](#)

Changes the definition of an event or its associated handler for automating predefined actions, or alters the definition of scheduled actions. You can also use this statement to hide the definition of an event handler.

[TRIGGER EVENT statement](#)

Triggers a named event. The event may be defined for event triggers or be a scheduled event.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP EXTERNLOGIN statement

Drops an external login from the database.

Syntax

- **Drop an external login**

DROP EXTERNLOGIN *login-name* **TO** *remote-server*

- **Drop an external login (include variables in syntax)**

DROP EXTERNLOGIN USER *string* | *variable* **SERVER** *string* | *variable*

Parameters

DROP clause Specify the local user login ID.

SERVER clause Specify the name of the remote server. The local user's alternate login name and password for that server is the external login that is deleted.

Remarks

DROP EXTERNLOGIN deletes an external login from the database.

Note For *required* parameters that accept variable names, the database server returns an error if any of the following conditions is true:

- The variable does not exist
- The contents of the variable are NULL
- The variable exceeds the length allowed by the parameter
- The data type of the variable does not match that required by the parameter

Privileges

You must have the MANAGE ANY USER system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example drops the DBA external login to the fictitious remote server, sybase1.

```
DROP EXTERNLOGIN DBA TO sybase1;
```

Related Information

[Dropping external logins \(SQL Central\)](#)

Delete external logins from users to remote servers and directory access servers that are no longer

required.

[CREATE EXTERNLOGIN statement](#)

Assigns an alternate login name and password to be used when communicating with a remote server.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP FUNCTION statement

Removes a function from the database.

Syntax

DROP FUNCTION [IF EXISTS] [*owner.*]*function-name*

Remarks

If you do not want an error returned when the DROP FUNCTION statement attempts to remove a function that does not exist, then use the IF EXISTS clause.

Privileges

You must be the owner of the function, or have the DROP ANY PROCEDURE or DROP ANY OBJECT system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Core Feature. The IF EXISTS clause is not in the standard.

Example

This example drops the fictitious function, MyFunction, from the database.

```
DROP FUNCTION MyFunction;
```

Related Information

[CREATE FUNCTION statement](#)

Creates a user-defined SQL function in the database.

[CREATE FUNCTION statement \[External call\]](#)

Creates an interface to a native or external function.

[CREATE FUNCTION statement \[Web service\]](#)

Creates a web client function that makes an HTTP or SOAP over HTTP request.

[ALTER FUNCTION statement](#)

Modifies a function.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP INDEX statement

Removes an index from the database.

Syntax

```
DROP INDEX [ IF EXISTS ] { [ [ owner.]table-name.]index-name | [ [ owner.]materialized-  
view-name. ]index-name }
```

Remarks

Use the IF EXISTS clause if you do not want an error returned when the DROP INDEX statement attempts to remove an index that does not exist.

When you specify the IF EXISTS clause and the named table cannot be located, an error is returned.

DROP INDEX is prevented when the statement affects an object that is currently being used by another connection.

The DROP INDEX statement cannot be executed when there are cursors opened with the WITH HOLD clause that use either statement or transaction snapshots.

Privileges

To drop an index on a table, you must be the owner of the table, or have one of the following privileges:

- REFERENCES privilege on the table
- DROP ANY INDEX system privilege
- DROP ANY OBJECT system privilege

To drop an index on a materialized view, you must be the owner of the materialized view, or have one of the following privileges:

- DROP ANY INDEX system privilege
- DROP ANY OBJECT system privilege

Side effects

Automatic commit. The DROP INDEX statement closes all cursors for the current connection.

If you use the DROP INDEX statement to drop an index on a local temporary table an error is returned indicating that the index could not be found. Use the DROP TABLE statement to drop a local temporary table. Indexes on local temporary tables are dropped automatically when the local temporary table goes out of scope.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

This example drops a fictitious index, MyIndex, from the database.

```
DROP INDEX MyIndex;
```

Related Information

[Snapshot isolation](#)

Snapshot isolation is designed to improve concurrency and consistency by maintaining different versions of data.

[CREATE INDEX statement](#)

Creates an index on a specified table or materialized view.

[ALTER INDEX statement](#)

Renames an index, primary key, or foreign key, or changes the clustered nature of an index.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP LDAP SERVER statement

Drops an LDAP server configuration object.

Syntax

```
DROP LDAP SERVER ldapua-server-name  
[ WITH DROP ALL REFERENCES ]  
[ WITH SUSPEND ]
```

Parameters

WITH DROP ALL REFERENCES clause Specify the DROP ALL REFERENCES clause to drop an LDAP server configuration object that is referenced from a login policy.

WITH SUSPEND clause Specify the WITH SUSPEND clause to drop an LDAP server configuration object that is in a READY or ACTIVE state.

Remarks

This statement removes the LDAP server configuration object from the SYSLDAPSERVER system view after checking that the LDAP server configuration object is not in the READY or ACTIVE state. The statement fails when the state is READY or ACTIVE to ensure that the LDAP server is not in active use. To override this check, specify the WITH SUSPEND clause.

By default, a reference to the LDAP server configuration object in a login policy also causes this statement to fail. To remove an LDAP server configuration object that is referenced in a login policy, add the DROP ALL REFERENCES clause. Adding DROP ALL REFERENCES does not remove the reference from the login policy; it allows you to drop the configuration object when there are references. You must still remove the reference to the LDAP server configuration object from the login policy.

Privileges

You must have the MANAGE ANY LDAP SERVER system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example suspends an LDAP server configuration object named apps_primary that is referenced in a login policy, and then drops it.

```
DROP LDAP SERVER apps_primary WITH DROP ALL REFERENCES WITH SUSPEND;
```

Related Information

[CREATE LDAP SERVER statement](#)

Creates an LDAP server configuration object.

[ALTER LDAP SERVER statement](#)

Alters an LDAP server configuration object.

[VALIDATE LDAP SERVER statement](#)

Validates an LDAP server configuration object.

[ALTER LOGIN POLICY statement](#)

Alters an existing login policy.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP LOGIN POLICY statement

Drops a login policy.

Syntax

DROP LOGIN POLICY *policy-name*

Parameters

policy-name The name of the login policy.

Remarks

The statement fails if you drop a policy that is assigned to a user. You cannot drop the root login policy. Use the ALTER USER statement to change a user's policy assignment.

Privileges

You must have the MANAGE ANY LOGIN POLICY system privilege.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example creates a login policy, Test11, and then deletes it.

```
CREATE LOGIN POLICY Test11;  
DROP LOGIN POLICY Test11;
```

Related Information

[Login policies](#)

A **login policy** consists of a set of rules that are applied when you create a database connection for a user.

[Deleting a login policy](#)

Delete any custom login policy that is not assigned to a user.

[ALTER LOGIN POLICY statement](#)

Alters an existing login policy.

[ALTER USER statement](#)

Alters user settings.

[COMMENT statement](#)

Stores a comment for a database object in the system tables.

[CREATE LOGIN POLICY statement](#)

Creates a login policy.

[CREATE USER statement](#)

Creates a database user or group.

[DROP USER statement](#)

Drops a user.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP MATERIALIZED VIEW statement

Removes a materialized view from the database.

Syntax

DROP MATERIALIZED VIEW [IF EXISTS] [*owner.*]*materialized-view-name*

Remarks

All data in the table is automatically deleted as part of the dropping process. All indexes and keys for the materialized view are dropped as well.

Use the IF EXISTS clause if you do not want an error returned when the DROP MATERIALIZED VIEW statement attempts to remove a materialized view that does not exist.

You cannot execute a DROP MATERIALIZED VIEW statement on an object that is currently being used by another connection.

Executing a DROP MATERIALIZED VIEW statement changes the status of all dependent regular views to INVALID. To determine view dependencies before dropping a materialized view, use the `sa_dependent_views` system procedure.

Privileges

You must be the owner of the materialized view, or have the DROP ANY MATERIALIZED VIEW or DROP ANY OBJECT system privilege.

Side effects

Automatic commit. If the materialized view had been populated, DROP MATERIALIZED VIEW will trigger an automatic checkpoint. Closes all cursors for the current connection.

When a view is dropped, all procedures and triggers are unloaded from memory, so that any procedure or trigger that references the view reflects the fact that the view does not exist. The unloading and loading of procedures and triggers can affect performance if you are regularly dropping and creating views.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example drops a fictitious materialized view, `MyMaterializedView`, from the database.

```
DROP MATERIALIZED VIEW MyMaterializedView;
```

Related Information

[Advanced: Status and properties for materialized views](#)

Materialized view availability and state can be determined from their status and properties.

[CREATE MATERIALIZED VIEW statement](#)

Creates a materialized view.

[ALTER MATERIALIZED VIEW statement](#)

Alters a materialized view.

[REFRESH MATERIALIZED VIEW statement](#)

Initializes or refreshes the data in a materialized view by executing its query definition.

[sa_dependent_views system procedure](#)

Returns the list of all dependent views for a given table or view.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

DROP MESSAGE statement

Removes a message from the database.

Syntax

DROP MESSAGE *msgnum*

Remarks

None.

Privileges

You must be owner, or have the DROP MESSAGE or DROP ANY OBJECT system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.
- **Transact-SQL** DROP MESSAGE supplies the functionality provided by the `sp_dropmessage()` system procedure in Adaptive Server Enterprise.

Example

The following example creates and then drops a new message. To run this example, you must also have the CREATE MESSAGE system privilege:

```
CREATE MESSAGE 20000 AS 'End of line reached';  
DROP MESSAGE 20000;
```

Related Information

[PRINT statement \[T-SQL\]](#)

Returns a message to the client, or display a message in the database server messages window.

[CREATE MESSAGE statement \[T-SQL\]](#)

Creates a message number/message string pair. The message number can be used in PRINT and RAISERROR statements.

[SYSUSERMESSAGE system view](#)

Each row in the SYSUSERMESSAGE system view holds a user-defined message for an error condition. The underlying system table for this view is ISYSUSERMESSAGE.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP MIRROR SERVER statement

Drops a mirror server.

Syntax

DROP MIRROR SERVER *mirror-server-name*

Remarks

Removes the specified mirror server definition from the database.

The mirror database stops. If the mirror database is the only database running on the server, then the server also stops.

Read-only scale-out and database mirroring each require a separate license.

Privileges

You must have the **MANAGE ANY MIRROR SERVER** system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

This example creates, and then drops, a mirror server named `scaleout_primary2`:

```
CREATE MIRROR SERVER "scaleout_primary2"  
  AS PRIMARY  
  connection_string = 'server=scaleout_primary1;host=winxp-2:6871,winxp-3:6872';  
DROP MIRROR SERVER "scaleout_primary2";
```

Related Information

[Database mirroring](#)

Database mirroring is a configuration of three database servers, running on separate computers, that co-operate to maintain copies of the database and transaction log files.

[Separately licensed components](#)

These components are licensed separately and may need to be ordered separately if not included in your edition of SQL Anywhere.

[CREATE MIRROR SERVER statement](#)

Creates or replaces a mirror server that is being used for database mirroring or read-only scale-out.

[ALTER MIRROR SERVER statement](#)

Modifies the attributes of a mirror server.

[COMMENT statement](#)

Stores a comment for a database object in the system tables.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

DROP MUTEX statement

Drops the specified mutex.

Syntax

```
DROP MUTEX [ IF EXISTS ] [ owner.]mutex-name
```

Parameters

- **owner** The owner of the mutex. *owner* can also be specified using an indirect identifier (for example, ``[@variable-name]``).
- **mutex-name** The name of the mutex. *semaphore-name* can also be specified using an indirect identifier (for example, ``[@variable-name]``).
- **IF EXISTS clause** Use this clause to drop a mutex only if it exists. If a mutex does not exist and this clause is specified, then nothing happens and no error is returned.

Remarks

If the mutex is locked by another connection, the drop operation proceeds without blocking but the mutex will persist in the namespace until the mutex is released. Connections waiting on the mutex receive an error immediately indicating that the object has been dropped.

Privileges

You must have the DROP ANY MUTEX SEMAPHORE or DROP ANY OBJECT system privilege, or be the owner of the mutex. For a temporary mutex, you must be the connection that created the mutex.

Side effects

Automatic commit, but only for permanent mutexes.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statement drops the protect_my_cr_section mutex:

```
DROP MUTEX protect_my_cr_section;
```

Related Information

[Mutexes and semaphores](#)

Use mutexes and semaphores in your application logic to achieve locking behavior and control and communicate the availability of resources.

[CREATE MUTEX statement](#)

Creates or replaces a mutex (lock) that can be used to lock a resource such as a file or a procedure.

[LOCK MUTEX statement](#)

Locks a resource such as a file or system procedure using a predefined mutex.

[RELEASE MUTEX statement](#)

Releases the specified connection-scope mutex, if it is locked by the current connection.

[SYSMUTEXSEMAPHORE system view](#)

Each row in the SYSMUTEXSEMAPHORE system view provides information about a user-defined mutex or semaphore in the database. The underlying system table for this view is ISYSMUTEXSEMAPHORE.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP ODATA PRODUCER statement

Drops an OData Producer.

Syntax

DROP ODATA PRODUCER [IF EXISTS] *name*

Remarks

Removes the definition of the OData Producer and all of its options from the system tables. Use this statement only when you want to delete the configuration data of the OData Producer. If you want to temporarily disable the OData Producer, then execute the following statement:

```
ALTER ODATA PRODUCER name NOT ENABLED;
```

Use the IF EXISTS clause if you do not want an error returned when the DROP ODATA PRODUCER statement attempts to remove an OData Producer that does not exist.

Privileges

You must have the MANAGE ODATA system privilege.

Side effects

If the OData server is running, then the specified producer is shut down.

Example

To drop the OData Producer named prod, execute the following statement:

```
DROP ODATA PRODUCER prod;
```

Related Information

[OData support](#)

OData (Open Data Protocol) enables data services over RESTful HTTP. It allows you to perform operations through URIs (Uniform Resource Identifiers) to access and modify information.

[Network protocol options](#)

Network protocol options enable you to work around traits of different network protocol implementations.

[ALTER ODATA PRODUCER statement](#)

Alters an OData Producer.

[CREATE ODATA PRODUCER statement](#)

Creates or replaces an OData Producer.

[COMMENT statement](#)

Stores a comment for a database object in the system tables.

[-xs database server option](#)

Specifies server-side web services communications protocols.

[SYSODATAPRODUCER system view](#)

Each row of the SYSODATAPRODUCER system view describes an OData Producer. The underlying

system table for this view is ISYSODATAPRODUCER.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP PROCEDURE statement

Removes a procedure from the database.

Syntax

DROP PROCEDURE [**IF EXISTS**] [*owner.*]*procedure-name*

Remarks

Use the IF EXISTS clause if you do not want an error returned when the DROP PROCEDURE statement attempts to remove a procedure that does not exist.

Privileges

You must be the owner of the procedure, or have the DROP ANY PROCEDURE or DROP ANY OBJECT system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Core Feature. The IF EXISTS clause is not in the standard.

Example

This example creates a procedure called NewDepartment, and then drops it. To run this example, you must also have the CREATE PROCEDURE privilege.

```
CREATE PROCEDURE NewDepartment(  
    IN id INT,  
    IN name CHAR(35),  
    IN head_id INT )  
BEGIN  
    INSERT  
    INTO GROUPO.Departments ( DepartmentID,  
        DepartmentName, DepartmentHeadID )  
    VALUES ( id, name, head_id );  
END;  
DROP PROCEDURE NewDepartment;
```

Related Information

[CREATE PROCEDURE statement](#)

Creates a user-defined SQL procedure in the database.

[CREATE PROCEDURE statement \[External call\]](#)

Creates an interface to a native or external procedure.

[CREATE PROCEDURE statement \[Web service\]](#)

Creates a user-defined web client procedure that makes HTTP or SOAP requests to an HTTP server.

[ALTER PROCEDURE statement](#)

Modifies a procedure.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP PUBLICATION statement [MobiLink] [SQL Remote]

Drops a publication.

Syntax

DROP PUBLICATION [**IF EXISTS**] [*owner.*] *publication-name*

owner, publication-name : identifier

Remarks

This statement is applicable only to MobiLink and SQL Remote.

In MobiLink, a publication identifies synchronized data in a SQL Anywhere remote database. In SQL Remote, publications identify replicated data in both consolidated and remote databases.

Use the IF EXISTS clause if you do not want an error returned when the DROP PUBLICATION statement attempts to remove a publication that does not exist.

DROP PUBLICATION requires exclusive access to all tables referred to in the publication.

Privileges

You must be the owner of the publication, or have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit. All subscriptions to the publication are dropped.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statement drops the pub_contact publication.

```
DROP PUBLICATION pub_contact;
```

Related Information

[Publications](#)

A publication is a database object that identifies the data that is to be synchronized. It defines the data to be uploaded, and it limits the tables that can be downloaded to. (The download is defined in the download_cursor script.)

[Dropping a publication](#)

You can drop a publication when it is no longer required.

[Dropping a publication \(SQL Central\)](#)

Drop a publication (all subscriptions to that publication are automatically deleted).

[ALTER PUBLICATION statement \[MobiLink\] \[SQL Remote\]](#)

Alters a publication. In MobiLink, a publication identifies synchronized data in a SQL Anywhere remote database. In SQL Remote, a publication identifies replicated data in both consolidated and remote databases.

[CREATE PUBLICATION statement](#) [\[MobiLink\]](#) [\[SQL Remote\]](#)

Creates a publication. In MobiLink, a publication identifies synchronized data in a SQL Anywhere remote database. In SQL Remote, publications identify replicated data in both consolidated and remote databases.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP REMOTE MESSAGE TYPE statement [SQL Remote]

Deletes a message type definition from a database.

Syntax

DROP REMOTE MESSAGE TYPE *message-system*

message-system :

FILE

| **FTP**

| **SMTP**

Remarks

The statement removes a message type from a database.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statement drops the FILE message type from a database.

```
DROP REMOTE MESSAGE TYPE FILE;
```

Related Information

[SQL Remote message systems](#)

In SQL Remote replication, a message system is a protocol for exchanging messages between the consolidated database and a remote database. SQL Remote exchanges data among databases using one or more underlying message systems.

[Deleting a message type \(SQL Central\)](#)

Delete message types, which removes the publisher address from the message definition.

[CREATE REMOTE \[MESSAGE\] TYPE statement \[SQL Remote\]](#)

Identifies a message link and return address for outgoing messages from a database.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP REMOTE CONNECTION statement

Drops remote data access connections to a remote server.

Syntax

```
DROP REMOTE CONNECTION TO server-name  
CLOSE { CURRENT | ALL | connection-id }  
[ FOR { EFFECTIVE USER | LOGIN USER | USER user-name } ]
```

Parameters

server-name The remote data access server that was specified in the CREATE SERVER statement.

CLOSE clause CLOSE CURRENT drops remote connections for the current local connection.

CLOSE ALL drops remote connections for all local connections.

CLOSE *connection-id* drops remote connections for the local connection with the specified ID.

FOR clause FOR EFFECTIVE USER drops remote connections that were created with the current effective user's externlogin credentials.

FOR LOGIN USER drops remote connections that were created with the current login user's externlogin credentials.

FOR USER *user-name* drops remote connections that were created with the externlogin credentials for *user-name*.

If the FOR clause is omitted, then remote connections for all users are dropped.

Remarks

The DROP REMOTE CONNECTION statement allows you to explicitly close connections to a remote server. You may find this useful when a remote connection becomes inactive or is no longer needed.

Privileges

You must have the SERVER OPERATOR system privilege.

Side effects

None

Example

Drop all remote connections, whether they are the current connection or not, to the myServer server for the effective user:

```
DROP REMOTE CONNECTION TO myServer CLOSE ALL FOR EFFECTIVE USER;
```

Drop the remote connection to the myServer server for the current local connection for user2:

```
DROP REMOTE CONNECTION TO myServer CLOSE CURRENT FOR USER user2;
```

Drop all remote connections, whether they are the current connection or not, to the myServer

server for user2:

```
DROP REMOTE CONNECTION TO myServer CLOSE ALL FOR USER user2;
```

Drop the remote connection to the myServer server for the current local connection with the current connection's login user:

```
DROP REMOTE CONNECTION TO myServer CLOSE CURRENT FOR LOGIN USER;
```

Drop the remote connection to the myServer server for the connection with the ID connection-id and the current effective user:

```
DROP REMOTE CONNECTION TO myServer CLOSE connectionId FOR EFFECTIVE USER;
```

Related Information

[CREATE SERVER statement](#)

Creates a remote server or a directory access server.

[ALTER SERVER statement](#)

Modifies the attributes of a remote server.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

DROP ROLE statement

Removes a role from the database, or converts a user-extended role back to a regular user.

Syntax

```
DROP ROLE [ FROM USER ] role-name  
[ WITH { REVOKE | DROP OBJECTS } ]
```

Parameters

- **role-name** Specify the name of the role you are dropping or converting.
- **FROM USER clause** Specify this clause to convert a user-extended role back to a regular user. The user retains any login privileges, system privileges, and roles they had.
- **WITH REVOKE clause** Specify WITH REVOKE when there are other users who have been granted *role-name*.
- **WITH DROP OBJECTS clause** Specify WITH DROP OBJECTS to drop the objects owned by *role-name*. If any of the objects cannot be dropped, for example because the object is currently in use, then the statement returns an error. You cannot specify this clause if *role-name* is a user-extended role.

Remarks

A user-defined role can be dropped from the database, and a user-extended role can be converted back to a regular user, as long as all dependent roles meet the minimum required number of administrative users with active passwords, as set by the `min_role_admin` database option.

When you convert a user-extended role back to a regular user, ownership of objects remains with the user that is being converted back to a regular user.

When you convert a user-extended role back to a regular user, any privileges that were granted to *role-name* remain with the user after they have been converted.

If you convert a user-extended role back to a regular user and any other roles and/or users were granted the user-extended role, the WITH REVOKE clause must be specified or else the statement returns an error message and fails.

If any objects impacted by the drop operation are in use, the statement returns an error message and the statement fails.

Privileges

You must have administrative rights for the role being dropped.

Side effects

Automatic commit

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statement converts a user-extended role named Joe back to a regular user. Objects

owned by the user-extended role are now owned by the regular user, Joe. Users or roles that had been granted Joe retain the underlying privileges associated with the role.

```
DROP ROLE FROM USER Joe;
```

The following statement drops a user-extended role named Jack from the database. If the role Jack owned any objects, ownership of the object reverts to user Jack. Users or roles that were granted Jack retain the underlying privileges associated with the role Jack.

```
DROP ROLE Jack;
```

The following statement converts a user-extended role named Sam back to a regular user. Users and roles who had been granted Sam have the privileges of Sam revoked.

```
DROP ROLE FROM USER Sam WITH REVOKE;
```

The following statement drops a role named Sales1. Users or roles that were granted Sales1 retain the underlying privileges associated with the Sales1.

```
DROP ROLE Sales1;
```

The following statement drops a role named Sales2. Users or roles that had been granted Sales2 lose all underlying privileges associated with Sales2.

```
DROP ROLE Sales2 WITH REVOKE;
```

The following statement converts a user-extended role named Marketing1 to a regular user named Marketing1, and drops any objects that it owned.

```
DROP ROLE FROM USER Marketing1 WITH DROP OBJECTS;
```

The following statement drops a role named Marketing2, drops the objects it owned, and revokes its underlying system privileges from those who had been granted the role.

```
DROP ROLE Marketing2 WITH REVOKE WITH DROP OBJECTS;
```

Related Information

[User-extended roles](#)

A **user-extended role** is a user that has been extended to be a role, and is a type of user-defined role.

[ALTER ROLE statement](#)

Migrates a compatibility role to a user-defined role, and then drops the compatibility role.

[CREATE ROLE statement](#)

Creates or replaces a role, creates a user-extended role, or modifies administrators for a role.

[min_role_admins option](#)

Sets the minimum number of administrators required to administer roles.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP SEMAPHORE statement

Drops a semaphore.

Syntax

```
DROP SEMAPHORE [ IF EXISTS ] [ owner.]semaphore-name
```

Parameters

- **owner** The owner of the semaphore. *owner* can also be specified using an indirect identifier (for example, ``[@variable-name]``).
- **semaphore-name** The name of the semaphore. *semaphore-name* can also be specified using an indirect identifier (for example, ``[@variable-name]``).
- **IF EXISTS clause** Use this clause to drop a semaphore only if it exists. If a semaphore does not exist and this clause is specified, then nothing happens and no error is returned.

Remarks

An error is returned to any connection that is waiting to decrement the semaphore.

Privileges

You must have the DROP ANY MUTEX SEMAPHORE or DROP ANY OBJECT system privilege, or be the owner of the semaphore. For a temporary semaphore, you must be the connection that created the mutex.

Side effects

Automatic commit, but only for permanent semaphores.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statement drops a semaphore called `license_counter`:

```
DROP SEMAPHORE license_counter;
```

Related Information

[Mutexes and semaphores](#)

Use mutexes and semaphores in your application logic to achieve locking behavior and control and communicate the availability of resources.

[CREATE SEMAPHORE statement](#)

Creates or replaces a semaphore and establishes the initial value for its counter. A semaphore is a locking mechanism that uses a counter to communicate and control the availability of a resource such as an external library or procedure.

[NOTIFY SEMAPHORE statement](#)

Increments the counter associated with a semaphore.

[WAITFOR SEMAPHORE statement](#)

Decrements the counter associated with a semaphore.

[SYSMUTEXSEMAPHORE system view](#)

Each row in the SYSMUTEXSEMAPHORE system view provides information about a user-defined mutex or semaphore in the database. The underlying system table for this view is ISYSMUTEXSEMAPHORE.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP SEQUENCE statement

Drops a sequence.

Syntax

DROP SEQUENCE [*owner.*] *sequence-name*

Remarks

If the named sequence cannot be located, an error message is returned. When you drop a sequence, all synonyms for the name of the sequence are dropped automatically by the database server.

Privileges

You must be the owner of the sequence, or have the DROP ANY SEQUENCE or DROP ANY OBJECT system privilege.

Side effects

None

Standards

- **ANSI/ISO SQL Standard** Sequences comprise optional ANSI/ISO SQL Language Feature T176.

Example

The following example creates and then drops a sequence named Test:

```
CREATE SEQUENCE Test
START WITH 4
INCREMENT BY 2
NO MAXVALUE
NO CYCLE
CACHE 15;
DROP SEQUENCE Test;
```

Related Information

[Use of a sequence to generate unique values](#)

You can use a **sequence** to generate values that are unique across multiple tables or that are different from a set of natural numbers.

[ALTER SEQUENCE statement](#)

Alters a sequence.

[CREATE SEQUENCE statement](#)

Creates a sequence that can be used to generate primary key values that are unique across multiple tables, and for generating default values for a table.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP SERVER statement

Drops a remote server from the catalog.

Syntax

- **Drop a remote server**

DROP [REMOTE] SERVER *server-name*

- **Drop a remote server (SAP HANA syntax)**

DROP REMOTE SOURCE *remote-source-name*

Parameters

The DROP SERVER, DROP REMOTE SERVER, and DROP REMOTE SOURCE statements are semantically equivalent. The DROP REMOTE SOURCE syntax is provided for compatibility with SAP HANA.

Remarks

You must drop all the proxy tables that have been defined for the remote server before this statement succeeds.

Privileges

You must have the SERVER OPERATOR system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example drops a fictitious server named ase_prod:

```
DROP SERVER ase_prod;
```

Related Information

[Remote data access](#)

Remote data access gives you access to data in other data sources as well as access to the files on the computer that is running the database server.

[Remote servers and remote table mappings](#)

SQL Anywhere presents tables to a client application as if all the data in the tables were stored in the database to which the application is connected. Before you can map remote objects to a local proxy table, you must define the remote server where the remote object is located.

[Directory access servers](#)

A **directory access server** is a remote server that gives you access to the local file structure of the

computer running the database server.

[ALTER SERVER statement](#)

Modifies the attributes of a remote server.

[CREATE SERVER statement](#)

Creates a remote server or a directory access server.

[DROP REMOTE CONNECTION statement](#)

Drops remote data access connections to a remote server.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP SERVICE statement

Drops a web service.

Syntax

DROP SERVICE *service-name*

Remarks

This statement deletes a web service listed in the ISYSWEBSERVICE system table.

Privileges

You must be the owner of the service, or have the MANAGE ANY WEB SERVICE system privilege.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following SQL statement drops a fictitious web service named WebServiceTable:

```
DROP SERVICE WebServiceTable;
```

Related Information

[CREATE SERVICE statement \[HTTP web service\]](#)

Creates a new HTTP web service.

[CREATE SERVICE statement \[SOAP web service\]](#)

Creates a new SOAP over HTTP or DISH service.

[ALTER SERVICE statement \[HTTP web service\]](#)

Alters an existing HTTP web service.

[ALTER SERVICE statement \[SOAP web service\]](#)

Alters an existing SOAP or DISH service over HTTP.

[SYSWEBSERVICE system view](#)

Each row in the SYSWEBSERVICE system view holds a description of a web service. The underlying system table for this view is ISYSWEBSERVICE.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP SPATIAL REFERENCE SYSTEM statement

Drops a spatial reference system.

Syntax

DROP SPATIAL REFERENCE SYSTEM [IF EXISTS] *name*

Remarks

Use the IF EXISTS clause if you do not want an error returned when the DROP SPATIAL REFERENCE SYSTEM statement attempts to remove a spatial reference system that does not exist.

Privileges

You must be the owner, or have the MANAGE ANY SPATIAL OBJECT or DROP ANY OBJECT system privilege.

Side effects

None

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example drops a fictitious spatial reference system named Test.

```
DROP SPATIAL REFERENCE SYSTEM Test;
```

Related Information

[Spatial data](#)

Spatial data is data that describes the position, shape, and orientation of objects in a defined space.

[CREATE SPATIAL REFERENCE SYSTEM statement](#)

Creates or replaces a spatial reference system.

[ALTER SPATIAL REFERENCE SYSTEM statement](#)

Changes the settings of an existing spatial reference system. See the Remarks section for considerations before altering a spatial reference system.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP SPATIAL UNIT OF MEASURE statement

Drops a spatial unit of measurement.

Syntax

DROP SPATIAL UNIT OF MEASURE [**IF EXISTS**] *identifier*

Remarks

Use the IF EXISTS clause if you do not want an error returned when the DROP SPATIAL UNIT OF MEASURE statement attempts to remove a spatial unit of measure that does not exist.

Privileges

You must be the owner of the spatial unit of measure, or have the MANAGE ANY SPATIAL OBJECT or DROP ANY OBJECT system privilege.

Side effects

None

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example drops a fictitious spatial unit of measure named Test.

```
DROP SPATIAL UNIT OF MEASURE Test;
```

Related Information

[Spatial data](#)

Spatial data is data that describes the position, shape, and orientation of objects in a defined space.

[CREATE SPATIAL UNIT OF MEASURE statement](#)

Creates or replaces a spatial unit of measurement.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP STATEMENT statement [ESQL]

Frees statement resources.

Syntax

DROP STATEMENT [*owner.*]*statement-name*

statement-name :

identifier

| *hostvar*

Remarks

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.

To drop the statement, you must first have prepared the statement.

Privileges

None.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard. In the ANSI/ISO SQL Standard, this functionality is provided by the DEALLOCATE PREPARE statement, which is part of the optional ANSI/ISO SQL Language Feature B032, "Extended dynamic SQL".

Example

The following are examples of DROP STATEMENT use:

```
EXEC SQL DROP STATEMENT S1;  
EXEC SQL DROP STATEMENT :stmt;
```

Related Information

[PREPARE statement \[ESQL\]](#)

Prepares a statement to be executed later, or defines a cursor.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP STATISTICS statement

Erases all column statistics on the specified columns.

Syntax

DROP STATISTICS [ON] [owner.]object-name [(column-list)]

object-name :

table-name

| *materialized-view-name*

| *temp-table-name*

Remarks

The database server optimizer uses column statistics to determine the best strategy for executing each statement. The database server automatically gathers and updates these statistics. Column statistics are stored permanently in the database in the ISYSCOLSTAT system table. Column statistics gathered while processing one statement are available when searching for efficient ways to execute subsequent statements.

Occasionally, the column statistics can become inaccurate or relevant statistics may be unavailable. This condition is most likely to arise when few queries have been executed since a large amount of data was added, updated, or deleted.

The DROP STATISTICS statement deletes all internal statistical data from the ISYSCOLSTAT system table for the specified columns. This drastic step leaves the optimizer with no access to essential statistical information. Without these statistics, the optimizer can generate inefficient data access plans, causing poor database performance.

The DROP STATISTICS statement requires an exclusive lock on the table against which it is being performed. Execution of the statement cannot proceed until all other connections that refer to the table have either committed or rolled back the referring transactions, or closed any open cursors that refer to the table.

This statement should be used only during problem determination or when reloading data into a database that differs substantially from the original data.

Privileges

You must be the table owner, or have the **MANAGE ANY STATISTICS** or **DROP ANY OBJECT** system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Related Information

[Optimizer estimates and statistics](#)

The optimizer chooses a strategy for processing a statement based on **column statistics** stored in the database and on **heuristics**.

[CREATE STATISTICS statement](#)

Recreates the column statistics used by the optimizer, and stores them in the ISYSCOLSTAT system table.

[SYSCOLSTAT system view](#)

The SYSCOLSTAT system view contains the column statistics, including histograms, that are used by the optimizer. The contents of this view are best retrieved using the `sa_get_histogram` stored procedure or the Histogram utility. The underlying system table for this view is ISYSCOLSTAT.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP SUBSCRIPTION statement [SQL Remote]

Drops a subscription for a user from a publication.

Syntax

```
DROP SUBSCRIPTION TO publication-name [ ( subscription-value ) ]  
FOR subscriber-id, ...
```

subscription-value : *string*

subscriber-id : *string*

Parameters

publication-name The name of the publication to which the user is being subscribed. This can include the owner of the publication.

subscription-value A string that is compared to the subscription expression of the publication. This value is required if and only if the subscription you are dropping contains a subscription value.

subscriber-id The user ID of the subscriber to the publication.

Remarks

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.

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.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statement drops a subscription for the SamS user ID to the pub_contact publication.

```
DROP SUBSCRIPTION TO pub_contact  
FOR Sam_Singer;
```

Related Information

[CREATE SUBSCRIPTION statement \[SQL Remote\]](#)

Creates a subscription for a user to a publication.

[STOP SUBSCRIPTION statement \[SQL Remote\]](#)

Stops a subscription for a user to a publication.

[SYSSUBSCRIPTION system view](#)

Each row in the SYSSUBSCRIPTION system view describes a subscription from one user ID (which must have the REMOTE system privilege) to one publication. The underlying system table for this view is ISYSSUBSCRIPTION.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

DROP SYNCHRONIZATION PROFILE statement [MobiLink]

Deletes a SQL Anywhere synchronization profile.

Syntax

DROP SYNCHRONIZATION PROFILE [IF EXISTS] *name*

Parameters

name The name of the synchronization profile to delete.

Remarks

Synchronization profiles are named collections of synchronization options that can be used to control synchronization. Use the IF EXISTS clause if you do not want an error returned when the DROP SYNCHRONIZATION PROFILE statement attempts to remove a synchronization profile that does not exist.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Related Information

[CREATE SYNCHRONIZATION PROFILE statement \[MobiLink\]](#)

Creates a SQL Anywhere synchronization profile.

[ALTER SYNCHRONIZATION PROFILE statement \[MobiLink\]](#)

Changes a SQL Anywhere synchronization profile. Synchronization profiles are named collections of synchronization options that can be used to control synchronization.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]

Drops a synchronization subscription in a remote database.

Syntax

```
DROP SYNCHRONIZATION SUBSCRIPTION { subscription-name |  
TO publication-name  
[ FOR ml-username, ... ] }
```

Parameters

subscription-name Specifies the name of the subscription to drop.

TO clause Specifies the name of a publication.

FOR clause Specifies one more users.

Omitting this clause drops the default settings for the publication.

Remarks

Requires exclusive access to all tables referred to in the publication.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example drops the subscription named sales:

```
DROP SYNCHRONIZATION SUBSCRIPTION sales;
```

The following example drops the subscription between the MobiLink user SSinger and the publication called sales_publication:

```
DROP SYNCHRONIZATION SUBSCRIPTION  
  TO user.sales_publication  
  FOR "SSinger";
```

The following example omits the FOR clause, and so drops the default settings for the publication called sales_publication:

```
DROP SYNCHRONIZATION SUBSCRIPTION  
  TO user.sales_publication;
```

Related Information

[Dropping MobiLink subscriptions](#)

You can drop a synchronization subscription for a MobiLink user if it is no longer required.

[ALTER SYNCHRONIZATION SUBSCRIPTION statement \[MobiLink\]](#)

Alters the properties of a synchronization subscription in a SQL Anywhere remote database.

[CREATE SYNCHRONIZATION SUBSCRIPTION statement \[MobiLink\]](#)

Creates a subscription in a SQL Anywhere remote database between a MobiLink user and a publication.

[SYSSYNC system view](#)

The SYSSYNC system view contains information relating to synchronization.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP SYNCHRONIZATION USER statement [MobiLink]

Drops one or more synchronization users from a SQL Anywhere remote database.

Syntax

DROP SYNCHRONIZATION USER *ml-username, ...*

ml-username : *identifier*

Remarks

Drop one or more synchronization users from a MobiLink remote database.

You must have exclusive access to all tables referred to by publications subscribed to by the user.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

All subscriptions associated with the user are also deleted.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

Remove MobiLink user SSinger from the database.

```
DROP SYNCHRONIZATION USER SSinger;
```

Related Information

[ALTER SYNCHRONIZATION USER statement \[MobiLink\]](#)

Alters the properties of a MobiLink user in a SQL Anywhere remote database.

[CREATE SYNCHRONIZATION USER statement \[MobiLink\]](#)

Creates a MobiLink user in a SQL Anywhere remote database.

[SYSSYNC system view](#)

The SYSSYNC system view contains information relating to synchronization.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP TABLE statement

Removes a table from the database.

Syntax

```
DROP TABLE [ IF EXISTS ] [ owner.]table-name
```

Remarks

When you remove a table, all data in the table is automatically deleted as part of the dropping process. All indexes and keys for the table are dropped as well.

Use the IF EXISTS clause if you do not want an error returned when the DROP TABLE statement attempts to remove a table that does not exist.

You cannot execute a DROP TABLE statement when the statement affects a table that is currently being used by another connection. Execution of a DROP TABLE statement is also prevented if there is a materialized view dependent on the table.

When you execute a DROP TABLE statement, the status of all dependent regular views change to INVALID. To determine view dependencies before dropping a table, use the `sa_dependent_views` system procedure.

Global temporary tables cannot be dropped unless all users that have referenced the temporary table have disconnected.

Privileges

You must be the owner of the table, or have the DROP ANY TABLE or DROP ANY OBJECT system privilege.

Side effects

Automatic commit. DROP TABLE may also cause an automatic checkpoint. Executing a DROP TABLE statement closes all cursors for the current connection.

You can use the DROP TABLE statement to drop a local temporary table.

Standards

- **ANSI/ISO SQL Standard** Core Feature, however, the IF EXISTS clause is not in the standard. The ability to drop a declared local temporary table with the DROP TABLE statement is not in the standard.

Example

This example drops the fictitious MyTable table from the database.

```
DROP TABLE MyTable;
```

This example drops the fictitious MyTable table from the database. Because IF EXISTS is specified, if the table does not exist, an error is *not* returned.

```
DROP TABLE IF EXISTS MyTable;
```


Related Information

[Dropping a table](#)

Use SQL Central to drop a table from your database, for example, when you no longer need it.

[CREATE TABLE statement](#)

Creates a new table in the database and, optionally, creates a table on a remote server.

[ALTER TABLE statement](#)

Modifies a table definition or disables dependent views.

[sa_dependent_views system procedure](#)

Returns the list of all dependent views for a given table or view.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP TEXT CONFIGURATION statement

Drops a text configuration object.

Syntax

DROP TEXT CONFIGURATION [*owner.*]*text-config-name*

Remarks

Attempting to drop a text configuration object with dependent text indexes results in an error. You must drop the dependent text indexes before dropping the text configuration object.

Text configuration objects are stored in the ISYTEXTCONFIG system table.

Privileges

You must be the owner of the text configuration object, or have the DROP ANY TEXT CONFIGURATION or DROP ANY OBJECT system privilege.

Side effects

Automatic commit

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statements create and drop the mytextconfig text configuration object:

```
CREATE TEXT CONFIGURATION mytextconfig FROM default_char;  
DROP TEXT CONFIGURATION mytextconfig;
```

Related Information

[Full text search](#)

You can perform full text searching on tables.

[Text configuration object concepts and reference](#)

A text configuration object controls what terms go into a text index when it is built or refreshed, and how a full text query is interpreted.

[DROP TEXT INDEX statement](#)

Removes a text index from the database.

[SYSTEXTCONFIG system view](#)

Each row in the SYSTEXTCONFIG system view describes one text configuration object, for use with the full text search feature. The underlying system table for this view is ISYTEXTCONFIG.

[CREATE TEXT CONFIGURATION statement](#)

Creates a text configuration object for use with building and updating text indexes.

[ALTER TEXT CONFIGURATION statement](#)

Alters a text configuration object.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP TEXT INDEX statement

Removes a text index from the database.

Syntax

```
DROP TEXT INDEX text-index-name  
ON [ owner. ] table-name
```

Parameters

ON clause Use this clause to specify the table or materialized view on which the text index was built.

Remarks

You must drop dependent text indexes before you can drop a text configuration object.

This statement cannot be executed when there are cursors opened with the WITH HOLD clause that use either statement or transaction snapshots.

Privileges

To drop a text index on a table, you must be the owner of the table, or have one of the following privileges:

- REFERENCES privilege on the table
- DROP ANY INDEX system privilege
- DROP ANY OBJECT system privilege

To drop a text index on a materialized view, you must be the owner of the materialized view, or have one of the following privileges:

- DROP ANY INDEX system privilege
- DROP ANY OBJECT system privilege

Side effects

Automatic commit

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statements create and drop the TextIdx text index:

```
CREATE TEXT INDEX TextIdx ON GROUP0.MarketingInformation ( Description )  
DROP TEXT INDEX TextIdx ON GROUP0.MarketingInformation;
```

Related Information

[Full text search](#)

You can perform full text searching on tables.

[Text index concepts and reference](#)

A text index stores positional information for terms in the indexed columns.

[Snapshot isolation](#)

Snapshot isolation is designed to improve concurrency and consistency by maintaining different versions of data.

[SYSTEXTCONFIG system view](#)

Each row in the SYSTEXTCONFIG system view describes one text configuration object, for use with the full text search feature. The underlying system table for this view is ISYSTEXTCONFIG.

[CREATE TEXT INDEX statement](#)

Creates a text index.

[ALTER TEXT INDEX statement](#)

Alters the definition of a text index.

[DROP TEXT INDEX statement](#)

Removes a text index from the database.

[REFRESH TEXT INDEX statement](#)

Refreshes a text index.

[TRUNCATE TEXT INDEX statement](#)

Deletes the data in a MANUAL or an AUTO REFRESH text index.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP TIME ZONE statement

Drops a time zone from the database.

Syntax

DROP TIME ZONE [IF EXISTS] *name*

Remarks

Use the IF EXISTS clause if you do not want an error returned when the DROP TIME ZONE statement attempts to remove a time zone that does not exist.

A time zone cannot be dropped if it has been set using the time_zone database option.

Privileges

You must have either the MANAGE TIME ZONE or DROP ANY OBJECT system privilege.

Side effects

Automatic commit.

Example

This example drops a fictitious time zone, MyTimeZone, from the database.

```
DROP TIME ZONE MyTimeZone;
```

Related Information

[ALTER TIME ZONE statement](#)

Modifies a time zone object.

[COMMENT statement](#)

Stores a comment for a database object in the system tables.

[CREATE TIME ZONE statement](#)

Creates a simulated time zone that can be used by any database.

[time_zone option](#)

Specifies which time zone the database uses for time zone calculations.

[SYSTIMEZONE system view](#)

Each row in the SYSTIMEZONE system view describes one time zone. The underlying system table for this view is ISYSTIMEZONE.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP TRACE EVENT statement

Drops a user-defined trace event.

Syntax

DROP TRACE EVENT [**IF EXISTS**] *trace-event-name*

Remarks

This statement only drops user-defined trace events. If you do not want an error returned when the DROP TRACE EVENT statement attempts to remove a trace event that does not exist, use the IF EXISTS clause. If one or more event tracing sessions reference the trace event, then the trace event cannot be dropped until all the referencing trace sessions are dropped.

System privileges

You must have the **MANAGE ANY TRACE SESSION** system privilege.

Side effects

None

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

Drop the trace event named `my_event`:

```
DROP TRACE EVENT my_event;
```

Related Information

[Event tracing](#)

Event tracing records information about system-defined and user-defined trace events to an event trace data (ETD) file.

[System events](#)

Each system event provides a hook on which you can program a set of actions.

[CREATE TEMPORARY TRACE EVENT statement](#)

Creates a user trace event that persists until the database is stopped.

[CREATE TEMPORARY TRACE EVENT SESSION statement](#)

Creates a user trace event session.

[ALTER TRACE EVENT SESSION statement](#)

Adds or removes trace events from a session, adds or removes targets from a session, or starts or stops a trace session.

[DROP TRACE EVENT SESSION statement](#)

Drops a trace event session.

[NOTIFY TRACE EVENT statement](#)

Logs a user-defined trace event to a trace session.

[sp_trace_events system procedure](#)

Returns information about the trace events in the database.

[sp_trace_event_fields](#) system procedure

Returns information about the fields of the specified trace event.

[sp_trace_event_session_events](#) system procedure

Lists the trace events that are part of a specific trace event session.

[sp_trace_event_session_targets](#) system procedure

Lists the targets of a trace session.

[sp_trace_event_session_target_options](#) system procedure

Lists the target options for a trace event session.

[Event Trace Data \(ETD\) File Management utility \(dbmanageetd\)](#)

Generates a diagnostic log that allows the user to examine information for user-defined and system events.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP TRACE EVENT SESSION statement

Drops a trace event session.

Syntax

```
DROP TRACE EVENT SESSION [ IF EXISTS ] session-name UNCONDITIONALLY  
[ ON SERVER ]
```

Remarks

When UNCONDITIONALLY is specified, dropping an active session automatically stops the session before removing its definition. Otherwise, an error is returned.

Specify the ON SERVER clause to drop a trace event session that is recording trace events from all databases on the database server. If this clause is not specified, then the trace event session on the current database is deleted.

System privileges

You must have the MANAGE ANY TRACE SESSION system privilege.

Side effects

None

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statement drops the trace event session named my_session:

```
DROP TRACE EVENT SESSION my_session;
```

Related Information

[Event tracing](#)

Event tracing records information about system-defined and user-defined trace events to an event trace data (ETD) file.

[System events](#)

Each system event provides a hook on which you can program a set of actions.

[CREATE TEMPORARY TRACE EVENT statement](#)

Creates a user trace event that persists until the database is stopped.

[CREATE TEMPORARY TRACE EVENT SESSION statement](#)

Creates a user trace event session.

[ALTER TRACE EVENT SESSION statement](#)

Adds or removes trace events from a session, adds or removes targets from a session, or starts or stops a trace session.

[DROP TRACE EVENT SESSION statement](#)

Drops a trace event session.

[NOTIFY TRACE EVENT statement](#)

Logs a user-defined trace event to a trace session.

[sp_trace_events system procedure](#)

Returns information about the trace events in the database.

[sp_trace_event_fields system procedure](#)

Returns information about the fields of the specified trace event.

[sp_trace_event_sessions system procedure](#)

Returns a list of the trace event sessions that are defined for the database.

[sp_trace_event_session_events system procedure](#)

Lists the trace events that are part of a specific trace event session.

[sp_trace_event_session_targets system procedure](#)

Lists the targets of a trace session.

[sp_trace_event_session_target_options system procedure](#)

Lists the target options for a trace event session.

[Event Trace Data \(ETD\) File Management utility \(dbmanageetd\)](#)

Generates a diagnostic log that allows the user to examine information for user-defined and system events.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP TRIGGER statement

Removes a trigger from the database.

Syntax

```
DROP TRIGGER [ IF EXISTS ] [ owner.] [ table-name.] trigger-name
```

Remarks

Use the IF EXISTS clause if you do not want an error returned when the DROP statement attempts to remove a database object that does not exist.

Privileges

To drop a trigger on a table, one of the following must be true:

- You are the owner of the table.
- You have ALTER privilege on the table.
- You have the ALTER ANY TABLE system privilege.
- You have the ALTER ANY OBJECT system privilege.

To drop a trigger on a view owned by someone else, you must have either the ALTER ANY VIEW or ALTER ANY OBJECT system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** DROP TRIGGER comprises part of optional ANSI/ISO SQL Language Feature T211, "Basic trigger capability". The IF EXISTS clause is not in the standard.

Example

This example creates, and then drops, a trigger called emp_upper_postal_code to ensure that postal codes are in upper case before updating the Employees table. If the trigger does not exist, an error is returned.

```
CREATE TRIGGER emp_upper_postal_code
BEFORE UPDATE OF PostalCode
ON GROUND.Employees
REFERENCING NEW AS new_emp
FOR EACH ROW
WHEN ( ISNUMERIC( new_emp.PostalCode ) = 0 )
BEGIN
    -- Ensure postal code is uppercase (employee might be
    -- in Canada where postal codes contain letters)
    SET new_emp.PostalCode = UPPER(new_emp.PostalCode)
END;
DROP TRIGGER MyTrigger;
```

Related Information

[Dropping a trigger](#)

Use SQL Central to drop a trigger from your database.

[CREATE TRIGGER statement](#)

Creates a trigger on a table.

[ALTER TRIGGER statement](#)

Replaces a trigger definition with a modified version. You must include the entire new trigger definition in the ALTER TRIGGER statement.

[ROLLBACK TRIGGER statement](#)

Undoes any changes made in a trigger.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP USER statement

Drops a user.

Syntax

DROP USER *user-name*

Parameters

user-name The name of the user you are dropping.

Remarks

Dropping a user also deletes all database objects (such as tables or procedures) that they own, as well as any external logins for the user. In addition, if the user is specified in the USER clause of any services, then those services are also dropped.

The user being removed cannot be connected to the database when the statement is executed.

If you use this statement in a procedure, do not specify the password as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

Privileges

You must have the MANAGE ANY USER system privilege.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example creates and then drops the user SQLTester.

```
CREATE USER SQLTester IDENTIFIED BY pass1234;  
DROP USER SQLTester;
```

Related Information

[Login policies](#)

A **login policy** consists of a set of rules that are applied when you create a database connection for a user.

[ALTER LOGIN POLICY statement](#)

Alters an existing login policy.

[ALTER USER statement](#)

Alters user settings.

[COMMENT statement](#)

Stores a comment for a database object in the system tables.

[CREATE LOGIN POLICY statement](#)

Creates a login policy.

[CREATE USER statement](#)

Creates a database user or group.

[DROP LOGIN POLICY statement](#)

Drops a login policy.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP VARIABLE statement

Drops a SQL variable.

Syntax

- **Drop a connection-scope variables:**

DROP VARIABLE [IF EXISTS] *identifier*

- **Drop a database-scope variables**

DROP DATABASE VARIABLE [IF EXISTS] [*owner.*]*identifier*

Parameters

identifier A valid identifier for the variable.

owner Specify the owner of the database-scope variable. If *owner* is not specified, the database server looks for a database-scope variable named *identifier* owned by the user executing the statement. If none is found, the database server looks for a database-scope variable named *identifier* owned by PUBLIC.

IF EXISTS clause Specify this clause to allow the statement to complete without returning an error if a variable with the specified name (and/or owner, if specified) is not found.

Remarks

The DROP VARIABLE statement drops a SQL variable.

Connection-scope variables are also automatically dropped when the database connection is terminated. Database-scope variables must be explicitly dropped.

If a statement is still accessing a database-scope variable at the time it is dropped, then the variable is still available in memory for that statement only.

Variables are often used for large objects, so dropping them after use or setting them to NULL can free up significant resources such as disk space and memory.

Privileges

Connection-scope variables: No privileges are required to drop a connection-scope variable.

Database-scope variables: No privileges are required to drop a self-owned database-scope variable. To drop a database-scope variable owned by another user or by PUBLIC, you must have the MANAGE ANY DATABASE VARIABLE system privilege.

Side effects

Connection-scope variables: No side effects are associated with dropping a connection-scope variable.

Database-scope variables: Dropping a database-scope variable causes an automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Related Information

[CREATE VARIABLE statement](#)

Creates a connection- or database-scope variable.

[SET statement](#)

Assigns a value to a SQL variable.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

DROP VIEW statement

Removes a view from the database.

Syntax

```
DROP VIEW [ IF EXISTS ] [ owner.]view-name
```

Remarks

Use the IF EXISTS clause if you do not want an error returned when the DROP VIEW statement attempts to remove a view that does not exist.

When you execute the DROP VIEW statement, the status of all dependent regular views change to INVALID. To determine view dependencies before dropping a view, use the `sa_dependent_views` system procedure.

If you execute a DROP VIEW statement on a view that has one or more INSTEAD OF triggers, an error is returned. You must drop the trigger before the view can be dropped or altered.

Privileges

You must be the owner of the view, or have the DROP ANY VIEW or DROP ANY OBJECT system privilege.

Side effects

Automatic commit. Executing a DROP VIEW statement closes all cursors for the current connection.

When a view is dropped, all procedures and triggers are unloaded from memory, so that any procedure or trigger that references the view reflects the fact that the view does not exist. The unloading and loading of procedures and triggers can affect performance if you are regularly dropping and creating views.

Standards

- **ANSI/ISO SQL Standard** Core Feature. The IF EXISTS clause is not in the standard.

Example

The following example creates a view called MyView, and then drops it. You must be able to select from the Employees table to execute the CREATE VIEW statement in the example.

```
CREATE VIEW MyView  
AS SELECT * FROM GROUP0.Employees;  
DROP VIEW MyView;
```

Related Information

[CREATE VIEW statement](#)

Creates a view on the database.

[ALTER VIEW statement](#)

Replaces a view definition with a modified version.

[sa_dependent_views system procedure](#)

Returns the list of all dependent views for a given table or view.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

EXCEPT statement

Returns the set difference of two query blocks.

Syntax

```
[ WITH temporary-views ] main-query-block
EXCEPT [ ALL | DISTINCT ] except-query-block
[ ORDER BY [ integer | select-list-expression-name ] [ ASC | DESC ], ... ]
[ FOR XML xml-mode ]
[ OPTION( query-hint, ... ) ]
```

query-hint :

MATERIALIZED VIEW OPTIMIZATION *option-value*

| **FORCE OPTIMIZATION**

| *option-name* = *option-value*

main-query-block : *query-block*

except-query-block : *query-block*

option-name : *identifier*

option-value :

hostvar (*indicator allowed*)

| *string*

| *identifier*

| *number*

Parameters

main-query-block A query block, comprising a SELECT statement or a query expression (possibly nested). Query blocks are explained in the documentation for comment elements in SQL.

except-query-block A query block, comprising a SELECT statement or a query expression (possibly nested). Query blocks are explained in the documentation for comment elements in SQL.

FOR XML clause This clause is defined in the documentation for the SELECT statement.

OPTION clause Use this clause to specify hints for executing the statement. The following hints are supported:

- **MATERIALIZED VIEW OPTIMIZATION** *option-value*
- **FORCE OPTIMIZATION**
- *option-name* = *option-value*. A **OPTION**(*isolation_level* = ...) specification in the query text overrides all other means of specifying isolation level for a query.

Remarks

The EXCEPT statement returns all rows in main-query-block except those that also appear in the except-query-block. Specify EXCEPT or EXCEPT DISTINCT if you do not want duplicates from

main-query-block to appear as duplicates in the result. Otherwise, specify EXCEPT ALL. Query blocks can be nested.

The use of EXCEPT alone is equivalent to EXCEPT DISTINCT.

The *main-query-block* and the *except-query-block* must be UNION-compatible; they must each have the same number of items in their respective SELECT lists, and the types of each expression should be comparable. If corresponding items in two SELECT lists have different data types, the database server chooses a data type for the corresponding column in the result and automatically convert the columns in each *query-block* appropriately.

EXCEPT ALL implements bag difference rather than set difference. For example, if *main-query-block* contains 5 (duplicate) rows with specific values, and *except-query-block* contains 2 duplicate rows with identical values, then EXCEPT ALL will return 3 rows.

The results of EXCEPT are the same as the results of EXCEPT ALL if *main-query-block* does not contain duplicate rows.

The column names displayed are the same column names that are displayed for the first *query-block* and these names are used to determine the expression names to be matched with the ORDER BY clause. An alternative way of customizing result set column names is to use a common table expression (the WITH clause).

Privileges

You must own the tables referenced in *query-block*, or have the SELECT ANY TABLE privilege.

Side effects

None

Standards

- **ANSI/ISO SQL Standard** EXCEPT DISTINCT is a Core Feature. EXCEPT ALL comprises the optional ANSI/ISO SQL Language Feature F304. Explicitly specifying the DISTINCT keyword with EXCEPT is optional ANSI/ISO SQL Language Feature T551. Specifying an ORDER BY clause with EXCEPT is ANSI/ISO SQL Language Feature F850. A *query-block* that contains an ORDER BY clause constitutes ANSI/ISO SQL Feature F851. A query block that contains a row-limit clause (SELECT TOP or LIMIT) comprises optional ANSI/ISO SQL Language Feature F857 or F858, depending on the context. The FOR XML clause and the OPTION clause are not in the standard.
- **Transact-SQL** EXCEPT is not supported by Adaptive Server Enterprise. However, both EXCEPT ALL and EXCEPT DISTINCT can be used in the Transact-SQL dialect supported by SQL Anywhere.

Related Information

[Common elements in SQL syntax](#)

Learn about language elements that are found in the syntax of many SQL statements.

[Set operators and the NULL value](#)

NULLs are treated differently by set operators UNION, EXCEPT, and INTERSECT than it is in search conditions.

[SELECT statement](#)

Retrieves information from the database.

[INTERSECT statement](#)

Computes the intersection between the result sets of two or more queries.

[UNION statement](#)

Combines the results of two or more SELECT statements or query expressions.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

EXECUTE IMMEDIATE statement [SP]

Enables dynamically constructed statements to be executed from within a procedure.

Syntax

EXECUTE IMMEDIATE [*execute-option*] *string-expression*

execute-option :

WITH QUOTES [**ON** | **OFF**]

| **WITH ESCAPES** { **ON** | **OFF** }

| **WITH BATCH** { **ON** | **OFF** }

| **WITH RESULT SET** { **ON** | **OFF** }

Parameters

WITH QUOTES clause When you specify WITH QUOTES or WITH QUOTES ON, any double quotes in the string-expression are assumed to delimit an identifier. When you do not specify WITH QUOTES, or specify WITH QUOTES OFF, the treatment of double quotes in the string-expression depends on the current setting of the quoted_identifier option.

WITH QUOTES is useful when an object name that is passed into the stored procedure is used to construct the statement that is to be executed, but the name might require double quotes and the procedure might be called when the quoted_identifier option is set to Off.

WITH ESCAPES clause WITH ESCAPES OFF causes any escape sequences (such as \n, \x, or \\) in the string-expression to be ignored. For example, two consecutive backslashes remain as two backslashes, rather than being converted to a single backslash. The default setting is equivalent to WITH ESCAPES ON.

One use of WITH ESCAPES OFF is for easier execution of dynamically constructed statements referencing file names that contain backslashes.

In some contexts, escape sequences in the *string-expression* are transformed before the EXECUTE IMMEDIATE statement is executed. For example, compound statements are parsed before being executed, and escape sequences are transformed during this parsing, regardless of the WITH ESCAPES setting. In these contexts, WITH ESCAPES OFF prevents further translations from occurring. For example:

```
BEGIN
  DECLARE String1 LONG VARCHAR;
  DECLARE String2 LONG VARCHAR;
  EXECUTE IMMEDIATE
    'SET String1 = ''One backslash: \\\\' ''';
  EXECUTE IMMEDIATE WITH ESCAPES OFF
    'SET String2 = ''Two backslashes: \\\\' ''';
  SELECT String1, String2
END
```

WITH BATCH clause The WITH BATCH clause allows you to control the execution of batches in EXECUTE IMMEDIATE statements. Setting WITH BATCH OFF provides protection against inadvertent SQL-injection when the procedure is run.

WITH BATCH ON is the default, except for procedures owned by dbo.

When WITH BATCH OFF is used, the statement specified by *string-expression* must be a single statement.

WITH RESULT SET clause The WITH RESULT SET clause allows the server to define correctly the procedure containing it. Specifying WITH RESULT SET ON or WITH RESULT SET OFF affects both what happens when the procedure is created, as well as what happens when the procedure is executed. The default option is WITH RESULT SET OFF.

You can have an EXECUTE IMMEDIATE statement return a result set by specifying WITH RESULT SET ON. With this clause, the containing procedure is marked as returning a result set.

Remarks

The EXECUTE IMMEDIATE statement extends the range of statements that can be executed from within procedures and triggers. It lets you execute dynamically prepared statements, such as statements that are constructed using the parameters passed in to a procedure.

Literal strings in the statement must be enclosed in single quotes. String literals cannot span multiple lines.

Only global variables can be referenced in a statement executed by EXECUTE IMMEDIATE.

Statements executed with EXECUTE IMMEDIATE do not have their plans cached.

Privileges

None.

Side effects

None. However, if the statement is a data definition statement with an automatic commit as a side effect, that commit does take place.

Standards

- **ANSI/ISO SQL Standard** EXECUTE IMMEDIATE is optional ANSI/ISO SQL Language Feature B031, "Basic dynamic SQL". The *execute-option* syntax is not in the standard. The ANSI/ISO SQL Standard prohibits the use of EXECUTE IMMEDIATE that returns a result set.

Example

The following procedure creates a table, where the table name is supplied as a parameter to the procedure.

```
CREATE PROCEDURE CreateTableProc(  
    IN tablename char(30)  
)  
  
BEGIN  
    EXECUTE IMMEDIATE  
    'CREATE TABLE ' || tablename ||  
    ' ( column1 INT PRIMARY KEY)'  
END;
```

To call the procedure and create a table called mytable:

```
CALL CreateTableProc( 'mytable' );
```

Related Information

[EXECUTE IMMEDIATE used in procedures, triggers, user-defined functions, and batches](#)

The EXECUTE IMMEDIATE statement allows statements to be constructed using a combination of literal strings (in quotes) and variables.

[Named parameters](#)

Functions and procedures that are referenced from the CALL statement, the EXECUTE statement (Transact-SQL), the FROM clause of a DML statement, and the TRIGGER EVENT statement support positional parameters and named parameters. Named parameters support specifying any subset of the available parameters in any order.

[quoted_identifier option](#)

Controls the interpretation of strings that are enclosed in double quotes.

[CREATE PROCEDURE statement](#)

Creates a user-defined SQL procedure in the database.

[CREATE PROCEDURE statement \[External call\]](#)

Creates an interface to a native or external procedure.

[CREATE PROCEDURE statement \[Web service\]](#)

Creates a user-defined web client procedure that makes HTTP or SOAP requests to an HTTP server.

[BEGIN statement](#)

Specifies a compound statement.

[EXECUTE statement \[ESQL\]](#)

Executes a prepared SQL statement.

[EXECUTE statement \[T-SQL\]](#)

Invokes a procedure (an Adaptive Server Enterprise-compatible alternative to the CALL statement), executes a prepared SQL statement in Transact-SQL.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

EXECUTE statement [ESQL]

Executes a prepared SQL statement.

Syntax

- **General use**

```
EXECUTE statement
[ USING { hostvar-list | [ SQL ] DESCRIPTOR sqlda-name } ]
[ INTO { into-hostvar-list | [ SQL ] DESCRIPTOR into-sqlda-name } ]
[ ARRAY :row-count ]
```

row-count : *integer* | *hostvar*

statement : *identifier* | *hostvar* | *string*

sqlda-name : *identifier*

into-sqlda-name : *identifier*

- **Execute immediately**

```
EXECUTE IMMEDIATE statement
```

statement : *string* | *hostvar*

Parameters

USING clause Results from a SELECT statement or a CALL statement are put into either the variables in the variable list or 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.

INTO clause If EXECUTE INTO is used with an INSERT statement, the inserted row is returned in the second descriptor. For example, when using auto-increment primary keys or 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 that was assigned to the row. The same thing can be achieved by using @@identity with auto-increment keys.

ARRAY clause 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 can improve performance. The integer value is the number of rows to be inserted. The SQLDA must contain a variable for each entry (number of rows * number of columns). The first row is placed in SQLDA variables 0 to (columns per row)-1, and so on. Similarly, the ARRAY clause can be used for wide updates and deletes using prepared UPDATE and DELETE statements.

Remarks

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.

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.

- **Execute** Execute the named dynamic statement, which was previously prepared. If the dynamic statement contains host variable placeholders that supply information for the request (bind variables), either the *sql-da-name* must specify a C variable which is a pointer to a SQLDA containing enough descriptors for all the bind variables occurring in the statement, or the bind variables must be supplied in the *hostvar-list*.
- **Execute immediately** 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, and is dropped on completion.

When performing wide insert, update and delete operations (ARRAY clause), host variables must be simple names and the rows of the wide update must be the same data type.

Privileges

The required privileges depend on the statement being executed.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** The EXECUTE statement comprises part of optional ANSI/ISO SQL Language Feature B031, "Basic dynamic SQL". The INTO clause is part of optional ANSI/ISO SQL Language Feature B032, "Extended dynamic SQL". The ARRAY clause is not in the standard.

The EXECUTE IMMEDIATE statement supported with Embedded SQL is also part of optional ANSI/ISO SQL Language Feature B031.

Example

This example executes a DELETE statement.

```
EXEC SQL EXECUTE IMMEDIATE
'DELETE FROM Employees WHERE EmployeeID = 105';
```

This example executes a prepared DELETE statement.

```
EXEC SQL PREPARE del_stmt FROM
'DELETE FROM Employees WHERE EmployeeID = :a';
EXEC SQL EXECUTE del_stmt USING :employee_number;
```

This example executes a prepared query.

```
EXEC SQL PREPARE sel1 FROM
'SELECT Surname FROM Employees WHERE EmployeeID = :a';
EXEC SQL EXECUTE sel1 USING :employee_number INTO :surname;
```

Related Information

[Cursors in Embedded SQL](#)

A cursor is used to retrieve rows from a query that has multiple rows in its result set.

[Named parameters](#)

Functions and procedures that are referenced from the CALL statement, the EXECUTE statement (Transact-SQL), the FROM clause of a DML statement, and the TRIGGER EVENT statement support positional parameters and named parameters. Named parameters support specifying any subset of the available parameters in any order.

[EXECUTE IMMEDIATE statement \[SP\]](#)

Enables dynamically constructed statements to be executed from within a procedure.

[PREPARE statement \[ESQL\]](#)

Prepares a statement to be executed later, or defines a cursor.

[DECLARE CURSOR statement \[ESQL\] \[SP\]](#)

Declares a cursor.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

EXECUTE statement [T-SQL]

Invokes a procedure (an Adaptive Server Enterprise-compatible alternative to the CALL statement), executes a prepared SQL statement in Transact-SQL.

Syntax

- **Call a stored procedure**

```
[ EXECUTE | EXEC ][ @return_status = ] [creator.]procedure_name [ argument, ... ]
```

argument :

```
[ @parameter-name = ] expression
```

```
| [ @parameter-name = ] @variable [ output ]
```

- **Execute dynamic statements within T-SQL stored procedures and triggers**

```
EXECUTE ( string-expression )
```

Remarks

The syntax for calling a stored procedure is implemented for Transact-SQL compatibility. EXECUTE calls a stored procedure, optionally supplying procedure parameters and retrieving output values and return status information. In Watcom SQL, use the CALL or EXECUTE IMMEDIATE statements.

With the syntax for executing statements, you can execute dynamic statements within Transact-SQL stored procedures and triggers. The EXECUTE statement extends the range of statements that can be executed from within procedures and triggers. It lets you execute dynamically prepared statements, such as statements that are constructed using the parameters passed in to a procedure. Literal strings in the statement must be enclosed in single quotes, and the statement must be on a single line. This syntax is implemented for Transact-SQL compatibility, but can be used in either Transact-SQL or Watcom SQL batches and procedures.

The Transact-SQL EXECUTE statement does not have a way to signify that a result set is expected. One way to indicate that a Transact-SQL procedure returns a result set is to include something like the following:

```
IF 1 = 0
    SELECT 1 AS a
```

You can also execute statements within Transact-SQL stored procedures and triggers.

Privileges

When calling a procedure, you must be the owner of the procedure, or have the EXECUTE ANY PROCEDURE system privilege.

When executing dynamic statements within T-SQL stored procedures and triggers, the required privileges depend on the statement being executed.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** The syntax for calling store procedures is not in the standard.

The syntax for executing dynamic statements offers functionality equivalent to the EXECUTE IMMEDIATE statement in the ANSI/ISO SQL Standard, which is optional ANSI/ISO SQL Language Feature B031, "Basic dynamic SQL". However, the syntax for executing dynamic statements differs from that of the ANSI/ISO SQL Standard.

Example

The following procedure illustrates how to execute a stored procedure.

```
CREATE PROCEDURE p1( @var INTEGER = 54 )
AS
PRINT 'on input @var = %1!', @var
DECLARE @intvar integer
SELECT @intvar=123
SELECT @var=@intvar
PRINT 'on exit @var = %1!', @var;
```

The following statement executes the procedure, supplying the input value of 23 for the parameter. If you are connected from an Open Client or JDBC application, the PRINT messages are displayed in the client window. If you are connected from an ODBC or Embedded SQL application, the messages are displayed in the database server messages window.

```
EXECUTE p1 23;
```

The following is an alternative way of executing the procedure, which is useful if there are several parameters.

```
EXECUTE p1 @var = 23;
```

The following statement executes the procedure, using the default value for the parameter

```
EXECUTE p1;
```

The following statement executes the procedure, and stores the return value in a variable for checking return status.

```
EXECUTE @status = p1 23;
```

Related Information

[Named parameters](#)

Functions and procedures that are referenced from the CALL statement, the EXECUTE statement (Transact-SQL), the FROM clause of a DML statement, and the TRIGGER EVENT statement support positional parameters and named parameters. Named parameters support specifying any subset of the available parameters in any order.

[CALL statement](#)

Invokes a procedure.

[EXECUTE statement \[ESQL\]](#)

Executes a prepared SQL statement.

[EXECUTE IMMEDIATE statement \[SP\]](#)

Enables dynamically constructed statements to be executed from within a procedure.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

EXIT statement [Interactive SQL]

Leaves Interactive SQL.

Syntax

```
{ EXIT | QUIT | BYE } [ return-code ]
```

return-code : *number* | *connection-variable*

Remarks

This statement closes the Interactive SQL window if you are running Interactive SQL as a windowed program, or terminates Interactive SQL altogether when running in command-prompt (batch) mode. In both cases, the database connection is also closed. Before closing the database connection, Interactive SQL automatically executes a COMMIT statement if the `commit_on_exit` option is set to On. If this option is set to Off, Interactive SQL performs an implicit ROLLBACK. By default, the `commit_on_exit` option is set to On.

The optional return code can be checked in batch files to determine the success or failure of the statements in an Interactive SQL script file. The default return code is 0.

Privileges

None.

Side effects

This statement automatically performs a commit if option `commit_on_exit` is set to On (the default); otherwise it performs an implicit rollback.

On Windows operating systems the optional return value is available as `ERRORLEVEL`.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example sets the Interactive SQL return value to 1 if there are any rows in table T, or to 0 if T contains no rows.

```
CREATE VARIABLE rowCount INT;  
CREATE VARIABLE retcode INT;  
SELECT COUNT(*) INTO rowCount FROM GROUP0.Products;  
IF( rowCount > 0 ) THEN  
    SET retcode = 1;  
ELSE  
    SET retcode = 0;  
END IF;  
EXIT retcode;
```

Note

You cannot write the following statement because EXIT is an Interactive SQL statement (not a SQL statement), and you cannot include any Interactive SQL statement in other SQL block statements.

```
CREATE VARIABLE rowCount INT;  
SELECT COUNT(*) INTO rowCount FROM T;  
IF( rowCount > 0 ) THEN  
    EXIT 1    // <-- not allowed  
ELSE  
    EXIT 0    // <-- not allowed  
END IF;
```

Related Information

[Interactive SQL](#)

Interactive SQL is a tool that lets you execute SQL statements, run SQL script files, as well as view and compare plans.

[SET OPTION statement](#)

Changes the values of database and connection options.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

EXPLAIN statement [ESQL]

Retrieves a text specification of the optimization strategy used for a particular cursor.

Syntax

```
EXPLAIN PLAN FOR CURSOR cursor-name  
{ INTO hostvar | USING DESCRIPTOR sqlda-name }
```

cursor-name : *identifier* | *hostvar*

sqlda-name : *identifier*

Remarks

The EXPLAIN statement retrieves a text representation of the optimization strategy for the named cursor. The cursor must be previously declared and opened.

The *hostvar* or *sqlda-name* variable must be of string type. The optimization string specifies in what order the tables are searched, and also which indexes are being used for the searches if any.

This string may be long, depending on the query, and has the following format:

```
table (index), table (index), ...
```

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 are accessed by the database server. After each table is a parenthesized index name. This is the index that is used to access the table. If no index is used (the table is scanned sequentially) the letters "seq" will appear for the index name. If a particular SQL SELECT statement involves subqueries, a colon (:) will separate each subquery's optimization string. These subquery sections will appear in the order that the database server executes the queries.

After successful execution of the EXPLAIN statement, the *sqlerrd* field of the SQLCA (SQLIOESTIMATE) is filled in with an estimate of the number of input/output operations required to fetch all rows of the query.

You can only execute this statement on cursors you opened.

Privileges

None.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example illustrates the use of EXPLAIN:


```
EXEC SQL BEGIN DECLARE SECTION;  
char plan[300];  
EXEC SQL END DECLARE SECTION;  
EXEC SQL DECLARE employee_cursor CURSOR FOR  
    SELECT EmployeeID, Surname  
    FROM Employees  
    WHERE Surname like :pattern;  
EXEC SQL OPEN employee_cursor;  
EXEC SQL EXPLAIN PLAN FOR CURSOR employee_cursor INTO :plan;  
printf( "Optimization Strategy: '%s'.n", plan );
```

The plan variable contains the following string:

```
'Employees <seq>'
```

Related Information

[Cursors in Embedded SQL](#)

A cursor is used to retrieve rows from a query that has multiple rows in its result set.

[The SQL Communication Area \(SQLCA\)](#)

The **SQL Communication Area (SQLCA)** is an area of memory that is used on every database request for communicating statistics and errors from the application to the database server and back to the application.

[DECLARE CURSOR statement \[ESQL\] \[SP\]](#)

Declares a cursor.

[PREPARE statement \[ESQL\]](#)

Prepares a statement to be executed later, or defines a cursor.

[FETCH statement \[ESQL\] \[SP\]](#)

Positions, or re-positions, a cursor to a specific row, and then copies expression values from that row into variables accessible from within the stored procedure or application.

[CLOSE statement \[ESQL\] \[SP\]](#)

Closes a cursor.

[OPEN statement \[ESQL\] \[SP\]](#)

Opens a previously declared cursor to access information from the database.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

FETCH statement [ESQL] [SP]

Positions, or re-positions, a cursor to a specific row, and then copies expression values from that row into variables accessible from within the stored procedure or application.

Syntax

- **Stored procedures**

```
FETCH [ cursor-position ] cursor-name  

INTO variable-list [ FOR UPDATE ]
```

- **Embedded SQL**

```
FETCH [ cursor-position ] cursor-name  

[ INTO { hostvar-list } ] USING [ SQL ] DESCRIPTOR sqlda-name ]  

[ PURGE ]  

[ BLOCK n ]  

[ FOR UPDATE ]  

[ ARRAY fetch-count ]
```

cursor-position :

```
NEXT | PRIOR | FIRST | LAST  

| { ABSOLUTE | RELATIVE } row-count
```

row-count : *number* | *hostvar*

cursor-name : *identifier* | *hostvar*

hostvar-list : [may contain indicator variables](#)

variable-list : [stored procedure variables](#)

sqlda-name : *identifier*

fetch-count : *integer* | *hostvar*

Parameters

INTO clause The INTO clause is optional. If it is not specified, the FETCH statement positions the cursor only. The *hostvar-list* is for Embedded SQL use only.

Note If *variable-list* only contains one item, and there are multiple items in the result set of the cursor, then *variable-list* is considered a row variable and the returned column values are assigned to the fields of the row variable in order.

cursor position An optional positional parameter allows the cursor to be moved before a row is fetched. If not specified, NEXT is assumed. If the fetch includes a positioning parameter and the position is outside the allowable cursor positions, the SQLE_NOTFOUND warning is issued and the SQLCOUNT field indicates the offset from a valid position.

The OPEN statement initially positions the cursor before the first row.

NEXT clause Next is the default positioning, and causes the cursor to advance one row before the row is fetched.

PRIOR clause Causes the cursor to back up one row before fetching.

RELATIVE clause 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 backward. So, 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.

ABSOLUTE clause The ABSOLUTE positioning parameter is used to go to a particular row. A zero indicates the position before the first row.

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 clause A short form for ABSOLUTE 1.

LAST clause A short form for ABSOLUTE -1.

Note

Inserts and some updates to DYNAMIC SCROLL cursors can cause problems with cursor positioning. The database server does not put inserted rows at a predictable position within a cursor unless there is an ORDER BY clause on the SELECT statement. Sometimes the inserted row does not appear until the cursor is closed and opened again.

This behavior occurs if a temporary table had to be created to open the cursor.

The UPDATE statement can 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).

BLOCK clause Rows may be fetched by the client application more than one at a time. This is referred to as block fetching, prefetching, or multi-row fetching. The first fetch causes several rows to be sent back from the database server. The client buffers these rows, and subsequent fetches are retrieved from these buffers without a new request to the database server.

The BLOCK clause is for use in Embedded SQL only. It gives the client and server a hint about how many rows may be fetched by the application. The special value of 0 means the request is sent to the database server and a single row is returned (no row blocking). The BLOCK clause will reduce the number of rows included in the next prefetch to the BLOCK value. To increase the number of rows prefetched, use the PrefetchRows connection parameter.

If you do not specify a BLOCK clause, the value specified on OPEN is used.

FETCH RELATIVE 0 always re-fetches the row.

If prefetch is disabled for the cursor, the BLOCK clause is ignored and rows are fetched one at a time. If ARRAY is also specified, then the number of rows specified by ARRAY are fetched.

PURGE clause The PURGE clause is for use in Embedded SQL only. It causes the client to flush its buffers of all rows, and then send the fetch request to the database server. This fetch request may return a block of rows.

FOR UPDATE clause 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 server to put an intent lock on the row. The lock is held until the end of the current transaction.

ARRAY clause The ARRAY clause is for use in Embedded SQL only. It allows so-called wide fetches, which retrieve more than one row at a time, and which may improve performance.

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.

Remarks

The FETCH statement retrieves one row from the named cursor. The cursor must have been previously opened.

Table reference variables (variables defined as type TABLE REF) are not supported for use in FETCH statements.

- **Embedded SQL use** The Embedded SQL FETCH statement does not support arrays.

A 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, the DECLARE statement actually generates code and must be executed before the FETCH statement.

The server returns in SQLCOUNT the number of records fetched, and always returns a SQLCOUNT greater than zero unless there is an error or warning.

If the SQLSTATE_NOTFOUND warning is returned on the fetch, the *sqlerrd*[2] field of the SQLCA (SQLCOUNT) contains the number of rows by which the attempted fetch exceeded the allowable cursor positions. 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 is positive if the attempted fetch was beyond the end of the cursor, and negative if the attempted fetch was before the beginning of the cursor. The cursor is positioned on the last row if the attempted fetch was beyond the end of the cursor, and on the first row if the attempted fetch was before the beginning of the cursor.

After successful execution of the fetch statement, the *sqlerrd*[1] field of the SQLCA (SQLIOCOUNT) is incremented by the number of input/output operations required to perform the fetch. This field is actually incremented on every database statement.

- **Single row fetch** One row from the result of the SELECT statement is put into the variables in the variable list. The correspondence is one-to-one from the SELECT list to the host variable list.
- **Multi-row fetch** One or more rows from the result of the SELECT statement are put into either the variables in *variable-list* or the program data areas described by *sqlda-name*. In either case, the correspondence is one-to-one from the SELECT list to either the *hostvar-list* or the *sqlda-name* descriptor array.

Privileges

The cursor must be opened and you must have the SELECT object-level privilege on the tables, or be owner of the tables referenced in the declaration of the cursor, or have the SELECT ANY TABLE system privilege.

Side effects

A FETCH statement may cause multiple rows to be retrieved from the server to the client if prefetching is enabled.

Standards

- **ANSI/ISO SQL Standard** With minor exceptions, the stored procedure syntax of the FETCH statement is a Core Feature of the ANSI/ISO SQL Standard. Scrolling options other than NEXT constitute optional ANSI/ISO SQL Language Feature F431, "Read-only scrollable cursors". The software does not support the optional FROM clause of the FETCH statement as documented in the Standard.

The Embedded SQL syntax is not in the standard.

The FOR UPDATE, PURGE, ARRAY, BLOCK, and USING [SQL] DESCRIPTOR clauses are not in the standard.

Example

The following is an Embedded SQL example:

```
EXEC SQL DECLARE cur_employee CURSOR FOR
SELECT EmployeeID, Surname FROM Employees;
EXEC SQL OPEN cur_employee;
EXEC SQL FETCH cur_employee
INTO :emp_number, :emp_name:indicator;
```

The following is a procedure example:

```
BEGIN
  DECLARE cur_employee CURSOR FOR
    SELECT Surname
    FROM Employees;
  DECLARE name CHAR(40);
  OPEN cur_employee;
  lp: LOOP
    FETCH NEXT cur_employee into name;
    IF SQLCODE <> 0 THEN LEAVE lp END IF;
    ...
  END LOOP;
  CLOSE cur_employee;
END
```

Related Information

[How locking works](#)

A lock is a concurrency control mechanism that protects the integrity of data during the simultaneous execution of multiple transactions.

[How to fetch data using Embedded SQL](#)

Fetching data in Embedded SQL is done using the SELECT statement.

[Wide fetches or array fetches using Embedded SQL](#)

The FETCH statement can be modified to fetch more than one row at a time, which may improve

performance. This is called a **wide fetch** or an **array fetch**.

[Cursors in Embedded SQL](#)

A cursor is used to retrieve rows from a query that has multiple rows in its result set.

[Cursors in procedures, triggers, user-defined functions, and batches](#)

Cursors retrieve rows one at a time from a query or stored procedure with multiple rows in its result set.

[SELECT statement](#)

Retrieves information from the database.

[DECLARE CURSOR statement \[ESQL\] \[SP\]](#)

Declares a cursor.

[FOR statement](#)

Repeats the execution of a statement list once for each row in a cursor.

[PREPARE statement \[ESQL\]](#)

Prepares a statement to be executed later, or defines a cursor.

[OPEN statement \[ESQL\] \[SP\]](#)

Opens a previously declared cursor to access information from the database.

[RESUME statement](#)

Resumes execution of a cursor that returns result sets.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

FOR statement

Repeats the execution of a statement list once for each row in a cursor.

Syntax

```
[ statement-label : ]
FOR for-loop-name AS cursor-name [ cursor-type ] CURSOR
  { FOR statement [ FOR { UPDATE [ cursor-concurrency ] | READ ONLY } ]
    | USING variable-name }
DO statement-list
END FOR [ statement-label ]
```

cursor-type :
NO SCROLL
 | **DYNAMIC SCROLL**
 | **SCROLL**
 | **INSENSITIVE**
 | **SENSITIVE**

cursor-concurrency : **BY** { **VALUES** | **TIMESTAMP** | **LOCK** }

variable-name : *identifier*

Parameters

NO SCROLL clause A cursor declared NO SCROLL is restricted to moving forward through the result set using FETCH NEXT and FETCH RELATIVE 0 seek operations.

As rows cannot be returned to once the cursor leaves the row, there are no sensitivity restrictions on the cursor. When a NO SCROLL cursor is requested, the database server supplies the most efficient kind of cursor, which is an asensitive cursor.

DYNAMIC SCROLL clause DYNAMIC SCROLL is the default cursor type. DYNAMIC SCROLL cursors can use all formats of the FETCH statement.

When a DYNAMIC SCROLL cursor is requested, the database server supplies an asensitive cursor. When using cursors there is always a trade-off between efficiency and consistency. Asensitive cursors provide efficient performance at the expense of consistency.

SCROLL clause A cursor declared SCROLL can use all formats of the FETCH statement. When a SCROLL cursor is requested, the database server supplies a value-sensitive cursor.

The database server must execute value-sensitive cursors in such a way that result set membership is guaranteed. DYNAMIC SCROLL cursors are more efficient and should be used unless the consistent behavior of SCROLL cursors is required.

INSENSITIVE clause A cursor declared INSENSITIVE has its values and membership fixed over its lifetime. The result set of the SELECT statement is materialized when the cursor is opened. FETCHING from an INSENSITIVE cursor does not see the effect of any other INSERT, UPDATE, MERGE, PUT, or DELETE statement from any connection, including the connection that opened the cursor.

SENSITIVE clause A cursor declared SENSITIVE is sensitive to changes to membership or

values of the result set.

FOR UPDATE clause FOR UPDATE is the default. Cursors default to FOR UPDATE for single-table queries without an ORDER BY clause, or if the `ansi_update_constraints` option is set to Off. When the `ansi_update_constraints` option is set to Cursors or Strict, then cursors over a query containing an ORDER BY clause default to READ ONLY. However, you can explicitly mark cursors as updatable using the FOR UPDATE clause.

FOR READ ONLY clause A cursor declared FOR READ ONLY cannot be used in UPDATE (positioned), DELETE (positioned), or PUT statements. Because it is expensive to allow updates over cursors with an ORDER BY clause or a join, cursors over a query containing a join of two or more tables are READ ONLY and cannot be made updatable unless the `ansi_update_constraints` database option is Off. In response to any request for a cursor that specifies FOR UPDATE, the database server provides either a value-sensitive cursor or a sensitive cursor. Insensitive and asensitive cursors are not updatable.

Remarks

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 *statement-list* once for each row in the cursor.

Valid cursor types include dynamic scroll (default), scroll, no scroll, sensitive, and insensitive.

The name and data type of each local variable is derived from the *statement* used in the cursor. With a SELECT statement, the data types are the data types of the expressions in the SELECT list. The names are the SELECT list item aliases, if they exist; otherwise, they are the names 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 are taken from the RESULT clause in the procedure definition.

The LEAVE statement can be used to resume execution at the first statement after the END FOR. If the ending *statement-label* is specified, it must match the beginning *statement-label*.

The cursor created by a FOR statement is implicitly opened WITH HOLD, so statements executed within the loop that cause a COMMIT do not cause the cursor to be closed.

Caution

If you do not specify *cursor-name*, *cursor-type* is used as *cursor-name*.

Privileges

None.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** The FOR statement is part of optional ANSI/ISO SQL Language Feature P002, "Computational completeness". The USING clause of the FOR statement is not in the standard. As with the DECLARE CURSOR statement, the use of *cursor-concurrency* is not in the standard, nor are the combinations of cursor sensitivity and cursor scrollability options.

Example

The following fragment illustrates the use of the FOR loop:

```
FOR names AS curs INSENSITIVE CURSOR FOR
SELECT Surname
FROM Employees
DO
    CALL search_for_name( Surname );
END FOR;
```

This fragment also illustrates the use of the FOR loop:

```
BEGIN
    FOR names AS curs SCROLL CURSOR FOR
        SELECT EmployeeID, GivenName FROM Employees where EmployeeID < 130
        FOR UPDATE BY VALUES
        DO
            MESSAGE 'emp: ' || GivenName;
        END FOR;
END
```

The following example shows the FOR loop being using inside of a procedure called myproc, which returns the top 10 employees from the Employees table, depending on the sort order specified when calling the procedure (asc for ascending, and desc for descending):

```
CALL sa_make_object( 'procedure', 'myproc' ) ;
ALTER PROCEDURE myproc (
    IN @order_by VARCHAR(20) DEFAULT NULL
)
RESULT ( Surname person_name_t )
BEGIN
    DECLARE @sql LONG VARCHAR;
    DECLARE @msg LONG VARCHAR;
    DECLARE LOCAL TEMPORARY TABLE temp_names( surnames person_name_t );
    SET @sql = 'SELECT TOP(10) * FROM Employees AS t ' ;

    CASE @order_by
    WHEN 'asc' THEN
        SET @sql = @sql || 'ORDER BY t.Surname ASC';
        SET @msg = 'Sorted ascending by last name: ';
    WHEN 'desc' THEN
        SET @sql = @sql || 'ORDER BY t.Surname DESC';
        SET @msg = 'Sorted ascending by last name: ';
    END CASE;

    FOR loop_name AS curs SCROLL CURSOR USING @sql
    DO
        INSERT INTO temp_names( surnames ) VALUES( Surname );
        MESSAGE( @msg || Surname ) ;
    END FOR;
    SELECT * FROM temp_names;
END ;
```

Calling the myproc procedure and specifying asc (for example, `CALL myproc('asc');`) returns the following results:

Surname
Ahmed
Barker
Barletta
Bertrand
Bigelow
Blaikie
Braun
Breault
Bucceri
Butterfield

Related Information

[Sensitive cursors](#)

Sensitive cursors can be used for read-only or updatable cursor types.

[Insensitive cursors](#)

These cursors have insensitive membership, order, and values. No changes made after cursor open time are visible.

[Value-sensitive cursors](#)

For value-sensitive cursors, membership is insensitive, and the order and value of the result set is sensitive.

[Asensitive cursors](#)

These cursors do not have well-defined sensitivity in their membership, order, or values. The flexibility that is allowed in the sensitivity permits asensitive cursors to be optimized for performance.

[DECLARE CURSOR statement \[ESQL\] \[SP\]](#)

Declares a cursor.

[FETCH statement \[ESQL\] \[SP\]](#)

Positions, or re-positions, a cursor to a specific row, and then copies expression values from that row into variables accessible from within the stored procedure or application.

[CONTINUE statement](#)

Restarts a loop.

[LOOP statement](#)

Repeats the execution of a statement list.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

FORWARD TO statement

Sends native syntax SQL statements to a remote server.

Syntax

- **Forward a single statement**

FORWARD TO *server-name sql-statement*

- **Enter passthrough mode**

FORWARD TO [*server-name*]

Parameters

server-name The name of the remote server.

sql-statement A command in the native SQL syntax of the remote server. The command or group of commands is enclosed in braces ({}) or single quotes.

Remarks

The FORWARD TO statement enables users to specify the server to which a passthrough connection is required. The statement can be used in two ways:

- **Syntax for forwarding a single statement** Send a single statement to a remote server.
- **Syntax for passthrough mode** Place the database server into passthrough mode for sending a series of statements to a remote server. All subsequent statements are passed directly to the remote server. To turn passthrough mode off, execute FORWARD TO without a *server-name* specification.

If you encounter an error from the remote server while in passthrough mode, you must still execute a FORWARD TO statement to turn passthrough off.

Proxy Tables: The user must login to connect to a remote server using this syntax. If the FORWARD TO statement appears in a procedure owned by a different owner, or if the statement appears in an event (or any procedure called by an event), no authentication has taken place and hence there is no password to be forwarded. The only way that proxy table usage can work in this scenario is to define an EXTERNLOGIN.

When establishing a connection to *server-name* on behalf of the user, the database server uses one of the following:

- A remote login alias set using CREATE EXTERNLOGIN
- If a remote login alias is not set up, the name and password used to communicate with the database server

If the connection cannot be made to the server specified, the reason is contained in a message returned to the user.

After statements are passed to the requested server, any results are converted into a form that can be recognized by the client program.

Note The FORWARD TO statement is a server directive and cannot be used in stored

procedures, triggers, events, or batches.

Privileges

None

Side effects

The remote connection is set to AUTOCOMMIT (unchained) mode for the duration of the FORWARD TO session. Any work that was pending before the FORWARD TO statement is automatically committed.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example sends a SQL statement to the remote server RemoteASE:

```
FORWARD TO RemoteASE { SELECT * FROM titles };
```

The following example shows a passthrough session with the remote server aseprod:

```
FORWARD TO aseprod;  
    SELECT * FROM titles;  
    SELECT * FROM authors;  
FORWARD TO;
```

Related Information

[PASSTHROUGH statement \[SQL Remote\]](#)

Starts or stops passthrough mode for SQL Remote administration.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)» [Alphabetical list of SQL statements](#)

FROM clause

Specifies the database tables or views involved in a DELETE, SELECT, or UPDATE statement. When used within a SELECT statement, the FROM clause can also be used in a MERGE or INSERT statement.

Syntax

FROM *table-expression*,...

table-expression :

table-name

| *view-name*

| *procedure-name*

| *derived-table*

| *lateral-derived-table*

| *pivoted-derived-table*

| *unpivoted-derived-table*

| *join-expression*

| **(** *table-expression*,... **)**

| *openstring-expression*

| *apply-expression*

| *contains-expression*

| *dml-derived-table*

| *openxml-operator*

| *array-operator*

table-name :

[*userid.*]*table-name*

[[**AS**] *correlation-name*]

[**WITH** (*hint* [...])]

[**FORCE INDEX** (*index-name*)]

view-name :

[*userid.*]*view-name* [[**AS**] *correlation-name*]

[**WITH** (*table-hint*)]

procedure-name :

[*owner.*]*procedure-name* ([*parameter*,...])

[**WITH** (*column-name data-type*,...)]

[[**AS**] *correlation-name*]

derived-table :

(*select-statement* **)**

[**AS**] *correlation-name* [**(** *column-name*,... **)**]

lateral-derived-table :

LATERAL (*select-statement* | *table-expression*)

[**AS**] *correlation-name* [**(** *column-name*,... **)**]

*pivoted-derived-table :**pivot-source-table* **PIVOT** [**XML**] (*pivot-clause*) [**AS**] *pivoted-correlation-name**unpivoted-derived-table :**unpivoted-derived-table :**unpivot-source-table* **UNPIVOT** [{ **INCLUDE** | **EXCLUDE** } **NULLS**] (*unpivot-clause*) [**AS**] *correlation-name**join-expression :**table-expression* *join-operator* *table-expression*[**ON** *join-condition*]*join-operator :*[**KEY** | **NATURAL**] [*join-type*] **JOIN**| **CROSS JOIN***join-type :***INNER**| **LEFT** [**OUTER**]| **RIGHT** [**OUTER**]| **FULL** [**OUTER**]*hint :**table-hint* | *index-hint**table-hint :***READPAST**| **UPDLOCK**| **XLOCK**| **FASTFIRSTROW**| **HOLDLOCK**| **NOLOCK**| **READCOMMITTED**| **READUNCOMMITTED**| **REPEATABLEREAD**| **SERIALIZABLE***index-hint :***NO INDEX**| **INDEX** ([**PRIMARY KEY** | **FOREIGN KEY**] *index-name* [, ..]) [**INDEX ONLY** { **ON** | **OFF** }]| **CLUSTERED INDEX** [**INDEX ONLY** { **ON** | **OFF** }]*openstring-expression :***OPENSTRING** ({ **FILE** | **VALUE** } *string-expression*)**WITH** (*rowset-schema*)[**OPTION** (*scan-option* ...)][**AS**] *correlation-name*

apply-expression :

table-expression { **CROSS** | **OUTER** } **APPLY** *table-expression*

contains-expression :

{ *table-name* | *view-name* } **CONTAINS** (*column-name* [...], *contains-query*) [[**AS**] *score-correlation-name*]

rowset-schema :

column-schema-list

| **TABLE** [*owner.*]*table-name* [(*column-list*)]

column-schema-list :

{ *column-name* *user-or-base-type* | **filler**() } [,...]

column-list :

{ *column-name* | **filler**() } [,...]

scan-option :

BYTE ORDER MARK { **ON** | **OFF** }

| **COMMENTS INTRODUCED BY** *comment-prefix*

| { **COMPRESSED** | **AUTO** | **NOT COMPRESSED** }

| **DELIMITED BY** *string*

| **ENCODING** *encoding*

| { **ENCRYPTED KEY** *key-expression* | **NOT ENCRYPTED** }

| **ESCAPE CHARACTER** *character*

| **ESCAPES** { **ON** | **OFF** }

| **FORMAT** { **TEXT** | **BCP** }

| **HEXADECIMAL** { **ON** | **OFF** }

| **QUOTE** *string*

| **QUOTES** { **ON** | **OFF** }

| **ROW DELIMITED BY** *string*

| **SKIP** *integer*

| **STRIP** { **ON** | **OFF** | **LTRIM** | **RTRIM** | **BOTH** }

key-expression : *string* | *variable*

contains-query : *string*

dml-derived-table :

(*dml-statement*) **REFERENCING** ([*table-version-names* | **NONE**])

dml-statement :

insert-statement

delete-statement

update-statement

merge-statement

table-version-names :

OLD [**AS**] *correlation-name* [**FINAL** [**AS**] *correlation-name*]

| **FINAL** [**AS**] *correlation-name*

Parameters

- **table-name** A base table or temporary table. Tables owned by a different user can be qualified by specifying the user ID. Tables owned by user-defined roles that the user is a grantee of are found by default without specifying the user ID.
- **view-name** Specifies a view to include in the query. As with tables, views owned by a different user can be qualified by specifying the user ID. Views owned by groups to which the current user belongs are found by default without specifying the user ID. Although the syntax permits table hints on views, these hints have no effect.
- **procedure-name** A stored procedure that returns a result set. This clause applies to the FROM clause of SELECT statements only. The parentheses following the procedure name are required even if the procedure does not take parameters. DEFAULT can be specified in place of an optional parameter.

The argument list can be specified by position or by using keyword format. By position, the arguments match up with the corresponding parameter in the parameter list for the procedure. By keyword, the arguments are matched up with the named parameters.

If the stored procedure returns multiple result sets, only the first one is used.

The WITH clause provides a way of specifying column name aliases for the procedure result set. If a WITH clause is specified, the number of columns must match the number of columns in the procedure result set, and the data types must be compatible with those in the procedure result set. If no WITH clause is specified, the column names and types are those defined by the procedure definition.

For Embedded SQL applications, when you create a procedure without a RESULT clause and the procedure returns a variable result set, a DESCRIBE of the SELECT statement referencing the procedure may fail. To prevent the failure of the DESCRIBE, include a WITH clause that describes the expected result set schema.

Selecting from a procedure generates a temporary table. For example, the following query creates a local temporary table:

```
BEGIN
...
    my: LOOP
        BEGIN
            SELECT TOP 1 NUMBER INTO conn_id FROM sa_conn_info( ) ORDER BY
NUMBER DESC;
        END;
    ...
    END LOOP my;
END
```

- **derived-table** You can supply a SELECT statement instead of table or view name in the FROM clause. A SELECT statement used in this way is called a derived table, and it must be given an alias. For example, the following statement contains a derived table, MyDerivedTable, which ranks products in the Products table by UnitPrice.

```

SELECT TOP 3 *
      FROM ( SELECT Description,
                    Quantity,
                    UnitPrice,
                    RANK() OVER ( ORDER BY UnitPrice ASC )
                        AS Rank
          FROM GPOU0.Products ) AS MyDerivedTable
ORDER BY Rank;

```

- **lateral-derived-table** A derived table, stored procedure, or joined table that may include references to objects in the parent statement (outer references). Use a lateral derived table to use an outer reference in the FROM clause.

Use outer references only to tables that precede the lateral derived table in the FROM clause. For example, you cannot use an outer reference to an item in the SELECT list.

The table and the outer reference must be separated by a comma. For example, the following queries are valid:

```

SELECT *
      FROM A, LATERAL( B LEFT OUTER JOIN C ON ( A.x = B.x ) ) myLateralDT;

```

```

SELECT *
      FROM A, LATERAL( SELECT * FROM B WHERE A.x = B.x ) myLateralDT;

```

```

SELECT *
      FROM A, LATERAL( procedure-name( A.x ) ) myLateralDT;

```

Specifying LATERAL (*table-expression*) is equivalent to specifying LATERAL (SELECT * FROM *table-expression*).

- **pivoted-derived-table** Pivots data in the FROM *table-name* expression into a pivoted derived table. The full syntax for this clause is described in the PIVOT clause topic.
- **unpivoted-derived-table** Unpivots data in the FROM *table-name* expression into an unpivoted derived table. The full syntax for this clause is described in the UNPIVOT clause topic.
- **openstring-expression** Specify an OPENSTRING clause to query within a file or a BLOB, treating the content of these sources as a set of rows. When doing so, you also specify information about the schema of the file or BLOB for the result set to be generated since you are not querying a defined structure such as a table or view. This clause applies to the FROM clause of a SELECT statement. It is not supported for UPDATE or DELETE statements.

The ROWID function is supported over the result set of a table generated by an OPENSTRING expression.

The following subclauses and parameters of the OPENSTRING clause are used to define and query data within files and BLOBs:

FILE and VALUE clauses Use the FILE clause to specify the file to query. Use the VALUE clause to specify the BLOB expression to query. The data type for the BLOB expression is assumed to be LONG BINARY. You can specify the READ_CLIENT_FILE function as a value to the VALUE clause.

If neither the FILE nor VALUE keyword is specified, VALUE is assumed.

When using FORMAT SHAPEFILE, only FILE is assumed.

WITH clause Use this clause to specify the rowset schema (column names and data types) of the data being queried. You can specify the columns directly (for example, `WITH (Surname CHAR(30), GivenName CHAR(30))`). You can also use the TABLE subclause to reference a table to use to obtain schema information from (for example, `WITH (TABLE dba.Employees (Surname, GivenName)`)). You must own or have SELECT privileges on the table you specify.

When specifying columns, you can specify filler() for columns that you want to skip in the input data (for example, `WITH (filler(), Surname CHAR(30), GivenName CHAR(30))`).

OPTION clause Use the OPTION clause to specify parsing options to use for the input file, such as escape characters, delimiters, encoding, and so on. Supported options comprise those options for the LOAD TABLE statement that control the parsing of an input file.

- **scan-option** For information about each scan option, see the load options for the LOAD TABLE statement.
- **apply-expression** Use this clause to specify a join condition where the right *table-expression* is evaluated for every row in the left *table-expression*. For example, you can use an apply expression to evaluate a function, procedure, or derived table for each row in a table expression.
- **contains-expression** Use the CONTAINS clause following a table name to filter the table and return only those rows matching the full text query specified with *contains-query*. Every matching row of the table is returned together with a score column that can be referred to by using *score-correlation-name*. If *score-correlation-name* is not specified, then the score column can be referred to by the default correlation name, contains.

With the exception of the optional correlation name argument, the CONTAINS clause takes the same arguments as the CONTAINS search condition.

There must be a text index on the columns listed in the CONTAINS clause.

The *contains-query* cannot be NULL or an empty string. If the text configuration settings cause all of the terms in the *contains-query* to be dropped, rows from the base table referenced by the *contains-expression* are not returned.

- **correlation-name** Use *correlation-name* to specify a substitute name for a table or view in the FROM clause. The substitute name can then be referenced from elsewhere in the statement. For example, emp and dep are correlation names for the Employees and Departments tables, respectively:

```
SELECT Surname, GivenName, DepartmentName
FROM GROUP0.Employees emp, GROUP0.Departments dep,
WHERE emp.DepartmentID=dep.DepartmentID;
```

Correlation names can be used to distinguish between different instances of the same table. For example, the following query joins the Employee table to itself, using the Mgr correlation name, to include the surname of each employee's manager in the result:

```

SELECT Emp.EmployeeID, Emp.Surname, Dept.DepartmentName, Mgr.Surname AS
ManagerName
FROM GROUP0.Employees AS Emp, GROUP0.Departments AS Dept,
GROUP0.Employees AS Mgr
WHERE Emp.DepartmentID = Dept.DepartmentID AND Emp.ManagerID =
Mgr.EmployeeID;

```

- **dml-statement** Use *dml-statement* to specify the DML statement (INSERT, DELETE, UPDATE, or MERGE) from which you want to select rows. During execution, the DML statement specified in *dml-derived-table* is executed first, and the rows affected by that DML are materialized into a temporary table whose columns are described by the REFERENCING clause. The temporary table represents the result set of *dml-derived-table*.

Use REFERENCING () or REFERENCING (NONE) if the results do not need to be materialized into a temporary table because you are not referencing them in the query.

If you specify REFERENCING () or REFERENCING (NONE), then the updated rows are not materialized into a temporary table that represents the result set of *dml-derived-table* because they are not being referenced in the query. The temporary table in this case is an empty table. Use this feature if you want *dml-statement* to be executed before the main statement is executed.

In the results, OLD columns contain the values as seen by the scan that finds the rows to include in the update operation. FINAL columns contain the values after referential integrity checks have been made, computed and default columns have been updated, and all triggers have been fired (excluding AFTER triggers of type FOR STATEMENT).

Statement	Supported table versions
INSERT	FINAL
DELETE	OLD
UPDATE	FINAL and/or OLD
MERGE	FINAL and/or OLD

When specifying both OLD and FINAL names, two correlation names are used; however, these are not true correlations since they both refer to the same result set. If you specify [REFERENCING \(OLD AS O FINAL AS F \)](#), then there is an implicit join predicate: `O.rowid = F.rowid`.

The INSERT statement only supports FINAL. Consequently, the values of updated rows that are modified by an INSERT ON EXISTING UPDATE statement do not appear in the result set of the derived table. Instead, use the MERGE statement to perform the insert-else-update processing.

The *dml-derived-table* statement can only reference one updatable table; updates over multiple tables return an error. Also, selecting from *dml-statement* is not allowed if the DML statement appears inside a correlated subquery or common table expression because the semantics of these constructs can be unclear.

- **openxml-operator** Use this parameter to return a result set from an XML document, by using the OPENXML operator.
- **array-operator** Use this parameter to return a result set from an ARRAY, by using an array operator such as UNNEST.
- **WITH table-hint clause** The WITH *table-hint* clause allows you to specify the behavior

to be used only for this table, and only for this statement. Use this clause to change the behavior without changing the isolation level or setting a database or connection option. Table hints can be used for base tables, temporary tables, and materialized views.

Caution

The WITH *table-hint* clause is an advanced feature that should be used only if needed, and only by experienced database administrators. In addition, the setting may not be respected in all situations.

- **Isolation level related table hints** Isolation level table hints are used to specify isolation level behavior when querying tables. They specify a locking method that is used only for the specified tables, and only for the current query. You cannot specify snapshot isolation levels as table hints.

Table hint	Description
HOLDLOCK	Sets the behavior to be equivalent to isolation level 3. This table hint is synonymous with SERIALIZABLE.
NOLOCK	Sets the behavior to be equivalent to isolation level 0. This table hint is synonymous with READUNCOMMITTED.
READCOMMITTED	Sets the behavior to be equivalent to isolation level 1.
READPAST	Instructs the database server to ignore, instead of block on, write-locked rows. This table hint can only be used with isolation level 1. The READPAST hint is respected only when the correlation name in the FROM clause refers to a base or globally shared temporary table. In other situations (views, proxy tables, and table functions) the READPAST hint is ignored. Queries within views may utilize READPAST as long as the hint is specified for a correlation name that is a base table. Using the READPAST table hint can lead to anomalies due to the interaction of locking and predicate evaluation within the server. In addition, you cannot use the READPAST hint against tables that are the targets of a DELETE, INSERT, or UPDATE statement.
READUNCOMMITTED	Sets the behavior to be equivalent to isolation level 0. This table hint is synonymous with NOLOCK.
REPEATABLEREAD	Sets the behavior to be equivalent to isolation level 2.

Table hint	Description
SERIALIZABLE	Sets the behavior to be equivalent to isolation level 3. This table hint is synonymous with HOLDLOCK.
UPDLOCK	Indicates that rows processed by the statement from the hinted table are locked using intent locks. The affected rows remain locked until the end of the transaction. UPDLOCK works at all isolation levels and uses intent locks.
XLOCK	Indicates that rows processed by the statement from the hinted table are to be locked exclusively. The affected rows remain locked until the end of the transaction. XLOCK works at all isolation levels and uses write locks.

Note

If you are writing queries for databases that participate in MobiLink synchronization, it is recommended that you do not use the READPAST table hint in your synchronization scripts.

If you are considering READPAST because your application performs many updates that affect download performance, an alternative solution is to use snapshot isolation.

- **Optimization table hint (FASTFIRSTROW)** The FASTFIRSTROW table hint allows you to set the optimization goal for the query without setting the optimization_goal option to First-row. When you use FASTFIRSTROW, the database server chooses an access plan that is intended to reduce the time to fetch the first row of the query's result.
- **WITH (*index-hint*) clause** The WITH (*index-hint*) clause allows you to specify index hints that override the query optimizer plan selection algorithms, and tell the optimizer exactly how to access the table using indexes. Index hints can be used for base tables, temporary tables, and materialized views.

- **NO INDEX** Use this clause to force a sequential scan of the table (indexes are not used). Sequential scans may be very costly.

- **INDEX ([PRIMARY KEY | FOREIGN KEY] *index-name* [,...])** Use this clause to specify up to four indexes that the optimizer must use to satisfy the query.

If any of the specified indexes cannot be used, an error is returned.

You can specify PRIMARY KEY or FOREIGN KEY to remove ambiguity in the cases where the PRIMARY KEY index and FOREIGN KEY index on a table have the same name.

If you specify an index name in the index hint without the PRIMARY or FOREIGN key, and multiple indexes with the same name exist on a table, the optimizer chooses the normal index. If a normal index does not exist, the optimizer chooses the primary key index. If a primary key index does not exist, the foreign key index is used instead.

index-name can be qualified by specifying the user ID and the table name of the index.

The indexes specified in the INDEX clause must be indexes defined for that table; otherwise, an error is returned. For example, `FROM Products WITH(INDEX (Products.xx))` returns an error if the index xx is not defined for the Products table. Likewise, `FROM Products WITH(INDEX (sales_order_items.sales_order_items))` returns an error because the sales_order_items.sales_order_items index exists but is not defined for the Products table.

- **INDEX ONLY { ON | OFF }** Use this clause to control whether an index-only retrieval of data is performed. If the INDEX (*index-name*...) clause is specified with INDEX ONLY ON, the database server attempts an index-only retrieval using the specified indexes. If any of the specified indexes cannot be used in satisfying an index-only retrieval, an error is returned (for example, if there are no indexes, or if the existing indexes cannot satisfy the query).

Specify INDEX ONLY OFF to prevent an index-only retrieval.

- **FORCE INDEX (*index-name*)** The FORCE INDEX (*index-name*) syntax is provided for compatibility, and does not support specifying more than one index. This clause is equivalent to `WITH (INDEX (index-name))`.
- **CLUSTERED INDEX** Use this clause to specify that the optimizer must use a clustered index if one exists. The index name is not specified as only one clustered index can exist for a base table. If a clustered index doesn't exist or it cannot be used, an error is returned.

Remarks

Subqueries and subselects are supported as arguments to stored procedures and table functions in the FROM clause. For example, the following FROM clause is valid:

```
SELECT *, ( SELECT 12 x ) D
FROM sa_rowgenerator( 1,( SELECT 12 x ) );
```

The SELECT, UPDATE, and DELETE statements require a table list to specify which tables are used by the statement.

Note Although the FROM clause description refers to tables, it also applies to views and derived tables unless otherwise noted.

The FROM clause 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.

You cannot use an ON phrase with CROSS JOIN.

Privileges

The FILE clause of *opening-string-expression* requires the READ FILE privilege.

The TABLE clause of *opening-string-expression* requires the user to own the referenced tables, or to have the SELECT ANY TABLE privilege.

Side effects

None.

Standards

ANSI/ISO SQL Standard The FROM clause is a fundamental part of the ANSI/ISO SQL Standard. The complexity of the FROM clause means that you should check individual components of a FROM clause against the appropriate portions of the standard. The following is a non-exhaustive list of optional ANSI/ISO SQL Language Features supported in the software:

- CROSS JOIN, FULL OUTER JOIN, and NATURAL JOIN constitute optional ANSI/ISO SQL Feature F401.
- INTERSECT and INTERSECT ALL constitute optional ANSI/ISO SQL Feature F302.
- EXCEPT ALL is optional ANSI/ISO SQL Language Feature F304.
- derived tables are ANSI/ISO SQL Language Feature F591.
- procedures in the FROM clause (table functions) are ANSI/ISO SQL Feature T326. The ANSI/ISO SQL Standard requires the keyword TABLE to identify the output of a procedure as a table expression, whereas in the software, the TABLE keyword is unnecessary.
- common table expressions are optional ANSI/ISO SQL Language Feature T121. Using a common table expression in a derived table nested within another common table expression is Language Feature T122.
- recursive table expressions are ANSI/ISO SQL feature T131. Using a recursive table expression in a derived table nested within a common table expression is optional ANSI/ISO SQL Language Feature T132.

The following components of the FROM clause are not in the standard:

- KEY JOIN.
- CROSS APPLY and OUTER APPLY.
- OPENSTRING.
- a *table-expression* using CONTAINS (full text search).
- specifying a *dml-statement* as a derived table.
- all table hints, including the use of WITH, FORCE INDEX, READPAST and isolation level hints.
- LATERAL (*table-expression*). LATERAL (*select-statement*) is in the ANSI/ISO SQL Standard as optional ANSI/ISO SQL Language Feature T491.

Example

The following are valid FROM clauses:

```
...
FROM GROUP0.Employees
...
```

```
...
FROM GROUP0.Employees NATURAL JOIN GROUP0.Departments
...
```



```

...
FROM GROUP0.Customers
KEY JOIN GROUP0.SalesOrders
KEY JOIN GROUP0.SalesOrderItems
KEY JOIN GROUP0.Products
...

...
FROM GROUP0.Employees CONTAINS ( Street, ' Way ' )
...

```

The following query illustrates how to use derived tables in a query:

```

SELECT Surname, GivenName, number_of_orders
FROM GROUP0.Customers JOIN
    ( SELECT CustomerID, COUNT(*)
      FROM GROUP0.SalesOrders
      GROUP BY CustomerID )
  AS sales_order_counts( CustomerID,
                        number_of_orders )
ON ( Customers.ID = sales_order_counts.CustomerID )
WHERE number_of_orders > 3;

```

The following query illustrates how to select rows from stored procedure result sets:

```

SELECT t.ID, t.QuantityOrdered AS q, p.name
FROM GROUP0.ShowCustomerProducts( 149 ) t JOIN GROUP0.Products p
ON t.ID = p.ID;

```

The following example illustrates how to perform a query by using the OPENSTRING clause to query a file. The CREATE TABLE statement creates a table called testtable with two columns, column1 and columns2. The UNLOAD statement creates a file called *testfile.dat* by unloading rows from the RowGenerator table. The SELECT statement uses the OPENSTRING clause in a FROM clause to query *testfile.dat* using the schema information from both the testtable and RowGenerator tables. The query returns one row with the value 49.

```

CREATE TABLE testtable( column1 CHAR(10), column2 INT );
UNLOAD SELECT * FROM RowGenerator TO 'testfile.dat';
SELECT A.column2
FROM OPENSTRING( FILE 'testfile.dat' )
WITH ( TABLE testtable( column2 ) ) A, RowGenerator B
WHERE A.column2 = B.row_num
AND A.column2 < 50
AND B.row_num > 48;

```

The following example illustrates how to perform a query using the OPENSTRING clause to query a string value. The SELECT statement uses the OPENSTRING clause in a FROM clause to query a string value using the schema information provided in the WITH clause. The query returns two columns with three rows.

```
SELECT *
FROM OPENSTRING( VALUE '1,"First"$2,"Second"$3,"Third"' )
WITH ( c1 INT, c2 VARCHAR(30))
OPTION ( DELIMITED BY ',' ROW DELIMITED BY '$' )
AS VALS
```

The following example illustrates how to perform a query to select the rows modified by a data manipulation statement. In this example, a warning is issued when the stock of blue items drops by more than half.

```
SELECT old_products.name, old_products.quantity, final_products.quantity
FROM
( UPDATE GROUP0.Products SET quantity = quantity - 10 WHERE color = 'Blue' )
REFERENCING ( OLD AS old_products FINAL AS final_products )
WHERE final_products.quantity < 0.5 * old_products.quantity;
```

The following query illustrates the use of the WITH clause when selecting from a fictitious procedure called ShowCustomerProducts:

```
SELECT sp.ident, sp.quantity, Products.name
FROM GROUP0.ShowCustomerProducts( 149 )
WITH ( ident INT, description CHAR(20), quantity INT ) sp
JOIN GROUP0.Products
ON sp.ident = Products.ID;
```

Related Information

[Text index concepts and reference](#)

A text index stores positional information for terms in the indexed columns.

[Row locks](#)

Row locks prevent lost updates and other types of transaction inconsistencies.

[What to specify when creating or altering text configuration objects](#)

There are many settings to configure when creating or altering a text configuration object.

[Example text configuration objects](#)

You can test how a text configuration object breaks a string into terms using the sa_char_terms and sa_nchar_terms system procedures.

[SELECT over a DML statement](#)

You can use a DML statement (INSERT, UPDATE, DELETE, or MERGE) as a table expression in a query FROM clause.

[Data manipulation statements](#)

The statements used to add, change, or delete data are called data manipulation statements, which are a subset of the data manipulation language (DML) statements part of ANSI SQL.

[Isolation levels and consistency](#)

You can control the degree to which the operations in one transaction are visible to the operations in other concurrent transactions by setting the **isolation level**.

[MobiLink isolation levels](#)

MobiLink connects to a consolidated database at the most optimal isolation level it can, given the isolation levels enabled on the RDBMS. The default isolation levels are chosen to provide the best performance while ensuring data consistency.

[The FROM clause: Specifying tables](#)

The FROM clause is required in every SELECT statement that returns data from tables, views, or stored procedures.

[Joins: Retrieving data from several tables](#)

To retrieve related data from more than one table, you perform a join operation using the SQL JOIN operator.

[%TYPE and %ROWTYPE attributes](#)

In addition to explicitly setting the data type for an object, you can also set the data type by specifying the %TYPE and %ROWTYPE attributes.

[PIVOT clause](#)

Pivots a table expression in the FROM clause of a SELECT statement (FROM *pivoted-derived-table*) into a pivoted derived table. Pivoted derived tables offer an easy way to rotate row values from a column in a table expression into multiple columns and perform aggregation where needed on the columns included in the result set.

[UNPIVOT clause](#)

Unpivots compatible-type columns of a table expression in a FROM clause (FROM *unpivoted-derived-table expression*) into rows in a derived table. Unpivoting is used to normalize data, for example when you have similar data stored in multiple columns in tables and you want to return it in one column.

[Named parameters](#)

Functions and procedures that are referenced from the CALL statement, the EXECUTE statement (Transact-SQL), the FROM clause of a DML statement, and the TRIGGER EVENT statement support positional parameters and named parameters. Named parameters support specifying any subset of the available parameters in any order.

[download_cursor table event](#)

A data script that defines a cursor to select rows to download and insert or update in the given table in the remote database.

[download_delete_cursor table event](#)

A data script that defines a cursor to select rows that are to be deleted in the remote database.

[upload_fetch table event](#)

A data script that fetches rows from a synchronized table in the consolidated database for row-level conflict detection.

[CONTAINS search condition](#)

Use the CONTAINS search condition to evaluate whether a value is contained in a set.

[optimization_goal option](#)

Determines whether query processing is optimized towards returning the first row quickly, or minimizing the cost of returning the complete result set.

[UNNEST array operator](#)

Creates a derived table from the given array expressions that results in one row per array element.

[OPENXML operator](#)

Generates a result set from an XML document.

[LOAD TABLE statement](#)

Imports bulk data into a database table from an external file.

[DELETE statement](#)

Deletes rows from the database.

[SELECT statement](#)

Retrieves information from the database.

[UPDATE statement](#)

Modifies rows in database tables.

[INSERT statement](#)

Inserts a single row or a selection of rows from elsewhere in the database into a table.

[MERGE statement](#)

Merges tables, views, and procedure results into a table or view.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

GET DATA statement [ESQL]

Gets string or binary data for one column of the current row of a cursor.

Syntax

```
GET DATA cursor-name  
COLUMN column-num  
OFFSET start-offset  
[ WITH TEXTPTR ]  
{ USING DESCRIPTOR sqlda-name | INTO hostvar, ... }
```

cursor-name : *identifier* | *hostvar*

column-num : *integer* | *hostvar*

start-offset : *integer* | *hostvar*

sqlda-name : *identifier*

Parameters

COLUMN clause The value of *column-num* starts at one, and identifies the column whose data is to be fetched. That column must be of a string or binary type.

OFFSET clause The *start-offset* 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.

WITH TEXTPTR clause If the WITH TEXTPTR clause is given, a text pointer is retrieved into a second host 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. The text pointer is a 16-bit binary value, and can be declared as follows:

```
DECL_BINARY( 16 ) textptr_var;
```

The WITH TEXTPTR clause can only be used with long data types (LONG BINARY, LONG VARCHAR, TEXT, IMAGE). If you attempt to use it with another data type, the error INVALID_TEXTPTR_VALUE is returned.

The total length of the data is returned in the SQLCOUNT field of the SQLCA structure.

USING DESCRIPTOR clause The *sqlda-name* specifies the SQLDA (SQL Descriptor Area) that receives the fetched data. The USING DESCRIPTOR clause provides a dynamic method of specifying host variables to receive fetched data.

INTO clause Use the INTO clause to specify the host variable that receives the fetched data. The indicator value for the target host variable is of type a_sql_len, which is currently a 16-bit value, so it is not always large enough to contain the number of bytes truncated. Instead, it contains a negative value if the field contains the NULL value, 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.

Similarly, if a LONG VARCHAR, LONG NVARCHAR, or LONG BINARY host variable is used with an

offset greater than zero, the `untrunc_len` field does not accurately indicate the size before truncation.

Remarks

Get a piece of one column value from the row at the current cursor position. The cursor must be opened and positioned on a row, using `FETCH`.

`GET DATA` is usually used to fetch `LONG BINARY` or `LONG VARCHAR` fields.

Privileges

None.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example uses `GET DATA` to fetch a binary large object (also called a BLOB).

```
EXEC SQL BEGIN DECLARE SECTION;
DECL_BINARY(1000) piece;
short ind;

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;
```

Related Information

[FETCH statement \[ESQL\] \[SP\]](#)

Positions, or re-positions, a cursor to a specific row, and then copies expression values from that row into variables accessible from within the stored procedure or application.

[READTEXT statement \[T-SQL\]](#)

Reads text and image values from the database, starting from a specified offset and reading a specified number of bytes. This feature is provided solely for compatibility with Transact-SQL and its use is not recommended.

[SET statement](#)

Assigns a value to a SQL variable.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

GET DESCRIPTOR statement [ESQL]

Retrieves information about a variable within a descriptor area, or retrieves its value.

Syntax

```
GET DESCRIPTOR descriptor-name  
{ hostvar = COUNT | VALUE { integer | hostvar } assignment, ... }
```

assignment :

hostvar =

TYPE

| **LENGTH**

| **PRECISION**

| **SCALE**

| **DATA**

| **INDICATOR**

| **NAME**

| **NULLABLE**

| **RETURNED_LENGTH**

descriptor-name : *identifier*

Remarks

The GET DESCRIPTOR statement is used to retrieve information about a variable within a descriptor area, or to retrieve its value.

The value { *integer* | *hostvar* } specifies the variable in the descriptor area about which the information is retrieved. Type checking is performed when doing GET...DATA to ensure that the host variable and the descriptor variable have the same data type. LONG VARCHAR and LONG BINARY are not supported by GET DESCRIPTOR...DATA.

If an error occurs, it is returned in the SQLCA.

Privileges

None.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** GET DESCRIPTOR is part of optional ANSI/ISO SQL Language Feature B031, "Basic dynamic SQL".

Example

The following example returns the type of the column with position col_num in sqlda.


```
int get_type( SQLDA *sqlda, int col_num )
{
    EXEC SQL BEGIN DECLARE SECTION;
    int ret_type;
    int col = col_num;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL GET DESCRIPTOR sqlda VALUE :col :ret_type = TYPE;
    return( ret_type );
}
```

Related Information

[The SQL descriptor area \(SQLDA\)](#)

The SQLDA (SQL Descriptor Area) is an interface structure that is used for dynamic SQL statements. The structure is used to pass information regarding host variables and SELECT statement results to and from the database. The SQLDA is defined in the header file *sqlda.h*.

[ALLOCATE DESCRIPTOR statement \[ESQL\]](#)

Allocates space for a SQL descriptor area (SQLDA).

[DEALLOCATE DESCRIPTOR statement \[ESQL\]](#)

Frees memory associated with a SQL descriptor area.

[SET DESCRIPTOR statement \[ESQL\]](#)

Describes the variables in a SQL descriptor area and to place data into the descriptor area.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

GET OPTION statement [ESQL]

Gets the current setting of an option. It is recommended that you use the CONNECTION_PROPERTY function instead.

Syntax

```
GET OPTION [ userid.]option-name  
{ INTO hostvar | USING DESCRIPTOR sqlda-name }
```

userid : *identifier*, *string* | *hostvar*

option-name :
identifier
| *string*
| *hostvar*

hostvar : *indicator variable* allowed

sqlda-name : *identifier*

Remarks

The GET OPTION statement is provided for compatibility with older versions of the software. The recommended way to get the values of options is to use the CONNECTION_PROPERTY system function.

The GET OPTION statement gets the option setting of the option *option-name* for the user *userid* or for the connected user if *userid* is not specified. This is either the user's personal setting or the PUBLIC 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.

If *option-name* does not exist, GET OPTION returns the warning SQLE_NOTFOUND.

Privileges

None.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statement illustrates use of GET OPTION.

```
EXEC SQL GET OPTION 'date_format' INTO :datefmt;
```

Related Information

[Alphabetical list of system procedures](#)

These are the system procedures that you can use to return data or perform operations on data.

[SET OPTION statement](#)

Changes the values of database and connection options.

[CONNECTION_PROPERTY function \[System\]](#)

Returns the value of a given connection property as a string.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

GOTO statement

Branches to a labeled statement.

Syntax

```
label-name:  
sql-statement(s)  
GOTO label-name
```

Remarks

Statements in a procedure, trigger, or batch can be labeled using a valid identifier followed by a colon (for example mylabel:), provided that the label is at the beginning of a loop, conditional, or block. The label can then be referenced in a GOTO statement, causing the execution point to move to the top of the loop/condition or the first statement within the block.

When referencing a label in a GOTO statement, do not specify the colon.

If you nest compound statements, then you can only go to labels within the current compound statement and any of its ancestor compound statements. You cannot go to labels located in other compound statements that are nested within the ancestors.

The database server supports the use of the GOTO statement in Transact-SQL procedures, triggers, or batches. In Transact-SQL, label use is not restricted to the beginning of loops, conditionals, or blocks; they can occur on any statement. However, the same restrictions apply to using the GOTO statement within nested compound statements.

Privileges

None.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

In the following example, if the GotoTest procedure is executed, then the GOTO lbl1 repositions execution to the SET i2 = 200 statement. The returned values for column i2 in the result are 203 for all 5 rows in the result set.

```

CREATE OR REPLACE PROCEDURE GotoTest()
RESULT ( id INT, i INT, i2 INT )
BEGIN
    DECLARE LOCAL TEMPORARY TABLE gotoTable( id INT DEFAULT AUTOINCREMENT, i INT,
i2 INT ) NOT TRANSACTIONAL;
    DECLARE i INT;
    DECLARE i2 INT;
    SET i = 100;
    lbl1: WHILE i < 105 LOOP
        SET i2 = 200;
        SET i2 = i2 + 1;
        lbl2: BEGIN
            SET i2 = i2 + 1;
            lbl3: BEGIN
                SET i2 = i2 + 1;
                INSERT INTO gotoTable(i, i2) VALUES(i, i2);
                SET i = i + 1;
                IF( i < 110 ) THEN
                    GOTO lbl1
                END IF;
            END lbl3;
        END lbl2;
    END LOOP;
    SELECT id, i, i2 FROM gotoTable ORDER BY id;
END;
CALL GotoTest();

```

Table 1: Results

id	i	i2
1	100	203
2	101	203
3	102	203
4	103	203
5	104	203

If the GotoTest procedure is changed to use GOTO lbl2 instead of GOTO lbl1, then the GOTO statement repositions execution to the SET i2 = i2 + 1 statement immediately after the lbl2: BEGIN statement, and the returned values in column i2 become 203, 205, 207, up to 221.

If the GotoTest procedure is changed to use GOTO lbl3, then the GOTO statement repositions execution to the SET i2 = i2 + 1 statement immediately after the lbl3: BEGIN statement, and the returned values in column i2 become 203, 204, 205, up to 212.

The following Transact-SQL batch prints the message "yes" in the database server messages window four times:

```
DECLARE @count SMALLINT
SELECT @count = 1
restart:
    PRINT 'yes'
    SELECT @count = @count + 1
    WHILE @count <=4
        GOTO restart;
```

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

GRANT statement

Grant system and object-level privileges to users and roles.

Syntax

- **Grant system privileges**

```
GRANT system-privilege [...]  
TO grantee [ ,... ]  
[ { WITH NO ADMIN | WITH ADMIN [ ONLY ] } OPTION ]
```

- **Grant object-level privileges**

```
GRANT object-level-privilege,...  
ON [ owner.]object-name  
TO to-userid,...  
[ WITH GRANT OPTION ]
```

object-level-privilege :

```
ALL [ PRIVILEGES ]  
| ALTER  
| DELETE  
| INSERT  
| REFERENCES [ ( column-name,... ) ]  
| SELECT [ ( column-name,... ) ]  
| UPDATE [ ( column-name,... ) ]
```

- **Grant the SET USER system privilege**

```
GRANT SET USER [ ( user-list | ANY [ WITH ROLES role-list ] ) ]  
TO grantee [...]  
[ { WITH NO ADMIN | WITH ADMIN [ ONLY ] } OPTION ]
```

- **Grant the CHANGE PASSWORD system privilege**

```
GRANT CHANGE PASSWORD [ ( user-list | ANY [ WITH ROLES role-list ] ) ]  
TO grantee [...]  
[ { WITH NO ADMIN | WITH ADMIN [ ONLY ] } OPTION ]
```

Parameters

- ***grantee*** The user ID of a user, or the name of a role. You cannot grant privileges to compatibility roles. You can grant privileges to any system role; however, only the following system roles support log ins: PUBLIC, dbo, diagnostics, rs_systabgroup, and SA_DEBUG
- ***object-level-privilege***
 - **ALL privilege** This privilege grants ALTER, DELETE, INSERT, REFERENCES, SELECT, and UPDATE privileges on tables. This privilege grants DELETE, INSERT, and UPDATE privileges on views.
 - **ALTER privilege** This privilege allows the user to alter the named table with the ALTER TABLE statement. This privilege is not allowed for views.
 - **DELETE privilege** This privilege allows the user to delete rows from the named

table or view.

- **INSERT privilege** This privilege allows the user to insert rows into the named table or view.
- **LOAD privilege** This privilege allows the user to load the named table.
- **REFERENCES privilege** This privilege allows the user to create indexes on the named table and on the foreign keys that reference the named tables. If column names are specified, the user can reference only those columns. REFERENCES privileges on columns cannot be granted for views, only for tables. INDEX is a synonym for REFERENCES.
- **SELECT privilege** This privilege allows the user to view information in the view or table. If column names are specified, the users are allowed to view only those columns. SELECT privileges on columns cannot be granted for views, only for tables.
- **TRUNCATE privilege** This privilege allows the user to truncate the named object.
- **UPDATE privilege** This privilege allows the user to update rows in the view or table. If column names are specified, the user can update only those columns.
- **WITH GRANT OPTION** If WITH GRANT OPTION is specified, then the named user ID is also given permission to GRANT the same privilege to other user IDs. Users who can exercise a role do not inherit the WITH GRANT OPTION if it is granted to a role.
- **FROM *internal-id*** This clause is for internal use only.
- **GRANT SET USER**
 - ***user-list*** Specify the comma-separated list of all user IDs (targets) users that *grantee-list* can impersonate. For example: `GRANT SET USER(u1, u2, u3)...`
 - **ANY [WITH ROLES *target_role-list*] clause** Specify who *grantee* can impersonate without providing specific user IDs.

If just ANY is specified, then the user can impersonate any other user. This is the default.

If ANY WITH ROLES *role-list* is specified, users in *grantee-list* can impersonate anyone who has at least one of the roles listed in *role-list*, where *role-list* is a comma-separated list of roles.
 - **WITH ADMIN [ONLY] OPTION option** The WITH ADMIN OPTION and WITH ADMIN ONLY OPTION clauses can only be specified with the ANY clause.
- **GRANT CHANGE PASSWORD**
 - ***user-list*** Specify a comma-separated list of users that *grantee* can change passwords for.
 - **ANY [WITH ROLES *role-list*]** Specify who *grantee* can change the password for without providing specific user IDs.

If just ANY is specified, then the user can change any user's password. This is the default.

If ANY WITH ROLES *role-list* is specified, the *grantee* can change the password for anyone who has at least one of the roles listed in *role-list*, where *role-list* is a comma-separated list of roles.
 - **WITH ADMIN [ONLY] OPTION option** The WITH ADMIN OPTION and WITH ADMIN ONLY OPTION clauses can only be specified with the ANY clause.

Remarks

You can grant privileges on disabled objects. Privileges on disabled objects are stored in the database and become effective when the object is enabled.

With the exception of the SYS role, you can grant/revoke additional privileges to/from a system role, provided you have administrative rights on the privileges you are granting/revoking.

Granting SET USER to a user multiple times, specifying different user IDs they can impersonate, grants additional users to the list they can impersonate (as opposed to overwriting the previous grants).

Granting impersonation rights (GRANT SET USER) is not an indication of whether a user can successfully impersonate another user. Evaluation of whether a user can impersonate another user is done when the user ID attempts to start impersonating another user by executing a SETUSER statement. The impersonating user must have the SET USER system privilege, and must meet the at-least criteria required for impersonation.

The GRANT syntax related to authorities, permissions, and groups used in pre-16.0 versions of the software is still supported but deprecated.

When granting system privileges to the MANAGE ROLES system privilege, you must use the special internal representation for MANAGE ROLES, which is SYS_MANAGE_ROLES_ROLE (for example, [GRANT privilege-name TO SYS_MANAGE_ROLES_ROLE;](#))

Privileges

You must have administrative rights for each privilege you grant.

To grant object-level privileges, you must also have the MANAGE ANY OBJECT PRIVILEGE system privilege, with administrative rights.

Side effects

None

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

Granting system privileges to roles To grant the CREATE ANY OBJECT system privilege to the role RoleA without giving RoleA administrative rights, execute the following statement:

```
GRANT CREATE ANY OBJECT TO RoleA;
```

To grant RoleA the CREATE ANY OBJECT system privilege along with the ability to grant or revoke the system privilege to and from other users and roles, execute the following statement:

```
GRANT CREATE ANY OBJECT TO RoleA WITH ADMIN OPTION;
```

To give RoleA administrative rights to the BACKUP DATABASE system privilege, but not the ability to use the BACKUP DATABASE privilege, execute the following statement:

```
GRANT BACKUP DATABASE TO RoleA WITH ADMIN ONLY OPTION;
```

Granting SET USER to users The following example specifies that User4 and User5 can

impersonate User1, User2, and User3.

```
GRANT SET USER ( User1, User2, User3 ) TO User4, User5;
```

The following example specifies that User1 can impersonate any user in the database. As well, User1 can grant the SET USER system privilege to other users.

```
GRANT SET USER (ANY) TO User1 WITH ADMIN OPTION;
```

The following example specifies that User1 can impersonate any user who has been granted the SYS_AUTH_BACKUP_ROLE compatibility role.

```
GRANT SET USER (ANY WITH ROLES SYS_AUTH_BACKUP_ROLE) TO User1;
```

Related Information

[Replication-related system roles](#)

There are two system roles related to replication: SYS_RUN_REPLICATION_ROLE, and SYS_REPLICATION_ADMIN_ROLE.

[Initiation of synchronization](#)

The client always initiates MobiLink synchronization. For SQL Anywhere clients, synchronization can be initiated using the dbmlsync utility, the dbmlsync API or the SQL SYNCHRONIZE statement. All share similar semantics but offer different interfaces to synchronization and different abilities to integrate synchronization with your own applications.

[System privileges](#)

There are many system privileges that can be granted.

[Roles](#)

There are three types of roles in the role-based security model: **system roles**, **user-defined roles** (which include user-extended roles), and **compatibility roles**.

[Compatibility roles](#)

Compatibility roles can be thought of as starter roles containing logical groups of privileges.

[User-defined roles](#)

A user-defined role is a collection you can create of system privileges, object-level privileges, and roles, typically created to group privileges related to a specific task or set of tasks.

[Impersonation](#)

A user can temporarily assume the identity of another user in the database, also known as **impersonation**, to perform operations, provided they have a superset of the privileges of the person they are impersonating.

[Changes in inheritance behavior for some authorities that became roles](#)

In pre-16.0 databases, if you granted the DBA, REMOTE DBA, BACKUP, RESOURCE AND VALIDATE authorities to a group, the underlying permissions were *not* inherited by members of the group.

[SETUSER statement](#)

Allows a user to assume the identity of (impersonate) another authorized user.

[GRANT statement \(authorities and groups\) \(deprecated\)](#)

Use the new GRANT statement for granting system privileges (previously permissions), and GRANT ROLE statement for granting roles (previously authorities and groups).

[REVOKE statement](#)

Revokes roles and privileges from users and roles.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

GRANT ROLE statement

Grant roles to users and roles.

Syntax

- **Grant system roles**

```
GRANT ROLE system-role
TO grantee [ ,... ]
```

system-role :

```
dbo
| DIAGNOSTICS
| PUBLIC
| rs_systabgroup
| SA_DEBUG
| SYS
| SYS_REPLICATION_ADMIN_ROLE
| SYS_RUN_REPLICATION_ROLE
| SYS_SAMONITOR_ADMIN_ROLE
| SYS_SPATIAL_ADMIN_ROLE
```

- **Grant user-defined roles**

```
GRANT ROLE user-defined-role [,...]
TO grantee [ ,... ]
[ { WITH NO ADMIN | WITH ADMIN [ ONLY ] } OPTION ]
```

- **Grant compatibility roles**

```
GRANT ROLE compatibility-role-name [,...]
TO grantee [ ,... ]
[ { WITH NO ADMIN | WITH ADMIN [ ONLY ] } OPTION ]
[ WITH NO SYSTEM PRIVILEGE INHERITANCE ]
```

compatibility-role-name :

```
SYS_AUTH_BACKUP_ROLE
| SYS_AUTH_DBA_ROLE
| SYS_AUTH_PROFILE_ROLE
| SYS_AUTH_READCLIENTFILE_ROLE
| SYS_AUTH_READFILE_ROLE
| SYS_AUTH_RESOURCE_ROLE
| SYS_AUTH_SA_ROLE
| SYS_AUTH_SSO_ROLE
| SYS_AUTH_VALIDATE_ROLE
| SYS_AUTH_WRITECLIENTFILE_ROLE
| SYS_AUTH_WRITEFILE_ROLE
```

Parameters

- ***role-name*** The name of a system role, compatibility role, user-extended role, or

user-defined role.

- **grantee** The user ID of a user, or the name of a role. You cannot grant roles to compatibility roles. You can grant roles to any system role; however, only the following system roles support log ins: PUBLIC, dbo, diagnostics, rs_systabgroup, and SA_DEBUG
- **WITH [NO] ADMIN OPTION clause** This clause is only applicable when granting non-system roles. You cannot grant administrative rights on a system role; only users with the MANAGE ROLES system privilege can administer (grant and revoke) system roles.

The default is WITH NO ADMIN OPTION, meaning that the *grantee* is given the role, but not the ability to administer it.

If WITH ADMIN OPTION is specified, each *grantee* is given administrative rights over each *role-granted*.

If WITH ADMIN ONLY OPTION is specified, then the *grantee* is given only administrative rights over the role, but is not given the role itself. You can never use the WITH NO SYSTEM PRIVILEGE INHERITANCE clause with the WITH ADMIN ONLY OPTION.

You can only use the WITH ADMIN OPTION clause with the WITH NO SYSTEM PRIVILEGE INHERITANCE clause when granting SYS_AUTH_DBA_ROLE.

- **WITH NO SYSTEM PRIVILEGE INHERITANCE** This clause prevents the grantees of a role from inheriting the role's system privileges. Normally, when you grant a compatibility role to a user or role, the compatibility role's system privileges are available to both the role and its grantees. When you disable the inheritance of a compatibility role's system privileges, the system privileges are available only to the role, not to its grantees.

The WITH NO SYSTEM PRIVILEGE INHERITANCE clause is provided for backwards compatibility. Disabling system privilege inheritance for a compatibility role mimics the behavior of the non-inheritable authority in version 12 and earlier databases. Enabling system privilege inheritance for a compatibility role mimics the behavior of all system roles and user-defined roles.

You can disable the inheritance of the system privileges when granting one of the following roles to users, user-extended roles, or system roles:

- SYS_AUTH_DBA_ROLE role
- SYS_AUTH_RESOURCE_ROLE
- SYS_AUTH_BACKUP_ROLE
- SYS_AUTH_VALIDATE_ROLE
- SYS_RUN_REPLICATION_ROLE

Also, you can only use the WITH ADMIN OPTION clause with the WITH NO SYSTEM PRIVILEGE INHERITANCE clause when granting SYS_AUTH_DBA_ROLE. You can never use the WITH NO SYSTEM PRIVILEGE INHERITANCE clause with the WITH ADMIN ONLY OPTION.

Disabling the system privilege inheritance for a user is only useful if you intend to convert the user to a user-extended role.

Remarks

With the exception of the SYS role, you can grant/revoke additional roles to/from a system role, provided you have administrative rights on the roles you are granting/revoking.

When granting a role to the MANAGE ROLES system privilege, you must use the special internal representation SYS_MANAGE_ROLES_ROLE. For example, `GRANT ROLE role-name TO`

`SYS_MANAGE_ROLES_ROLE;`

The GRANT syntax related to authorities, permissions, and groups used in pre-16.0 versions of the software is still supported but deprecated.

Privileges

You must have administrative rights for each role you grant.

To grant the SYS_REPLICATION_ADMIN_ROLE system role, you must have the MANAGE ROLES system privilege.

To grant the SYS_REPLICATION_RUN_ROLE system role, you must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

None

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

Granting roles to users To grant user Bob the role called SecurityRole without administrative rights, execute the following statement:

```
GRANT ROLE SecurityRole TO Bob;
```

To grant the role RoleB all the privileges associated with RoleA but no administrative rights for RoleA, execute the following statement:

```
GRANT ROLE RoleA TO RoleB;
```

To grant RoleB along with its administrative rights to the user Jane, execute the following statement:

```
GRANT ROLE RoleB TO Jane WITH ADMIN OPTION;
```

To grant the user John the administrative rights to RoleB, but the inability to use RoleB, execute the following statement:

```
GRANT ROLE RoleB TO John WITH ADMIN ONLY OPTION;
```

Granting the SYS_RUN_REPLICATION_ROLE system role The following example grants the SYS_RUN_REPLICATION_ROLE system role to grantee Sam_Singer:

```
GRANT ROLE SYS_RUN_REPLICATION_ROLE  
TO Sam_Singer;
```

Granting the SYS_REPLICATION_ADMIN_ROLE system role The following example grants the SYS_REPLICATION_ADMIN_ROLE role to grantee Sam_Singer:

```
GRANT ROLE SYS_REPLICATION_ADMIN_ROLE  
TO Sam_Singer;
```

Related Information

[Role administrators](#)

Role administrators are responsible for granting and revoking user-defined roles to users and other roles.

[Initiation of synchronization](#)

The client always initiates MobiLink synchronization. For SQL Anywhere clients, synchronization can be initiated using the dbmlsync utility, the dbmlsync API or the SQL SYNCHRONIZE statement. All share similar semantics but offer different interfaces to synchronization and different abilities to integrate synchronization with your own applications.

[System privileges](#)

There are many system privileges that can be granted.

[Roles](#)

There are three types of roles in the role-based security model: **system roles**, **user-defined roles** (which include user-extended roles), and **compatibility roles**.

[Compatibility roles](#)

Compatibility roles can be thought of as starter roles containing logical groups of privileges.

[User-defined roles](#)

A user-defined role is a collection you can create of system privileges, object-level privileges, and roles, typically created to group privileges related to a specific task or set of tasks.

[Impersonation](#)

A user can temporarily assume the identity of another user in the database, also known as **impersonation**, to perform operations, provided they have a superset of the privileges of the person they are impersonating.

[Replication-related system roles](#)

There are two system roles related to replication: SYS_RUN_REPLICATION_ROLE, and SYS_REPLICATION_ADMIN_ROLE.

[Changes in inheritance behavior for some authorities that became roles](#)

In pre-16.0 databases, if you granted the DBA, REMOTE DBA, BACKUP, RESOURCE AND VALIDATE authorities to a group, the underlying permissions were *not* inherited by members of the group.

[SETUSER statement](#)

Allows a user to assume the identity of (impersonate) another authorized user.

[GRANT statement \(authorities and groups\) \(deprecated\)](#)

Use the new GRANT statement for granting system privileges (previously permissions), and GRANT ROLE statement for granting roles (previously authorities and groups).

[REVOKE ROLE statement](#)

Revokes roles and privileges from users and roles.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)

» [Alphabetical list of SQL statements](#)

GRANT CONNECT statement

Creates a new user, and can also be used by a user to change their own password. However, it is recommended that you use the CREATE USER statement to create users instead of the GRANT CONNECT statement.

Syntax

```
GRANT CONNECT TO userid, ...  
[ IDENTIFIED BY password, ... ]
```

Parameters

- **CONNECT TO clause** Creates a new user. GRANT CONNECT can also be used by any user to change their own password. To create a user with an empty string as the password, use:

```
GRANT CONNECT TO userid IDENTIFIED BY "";
```

To create a user with no password, use:

```
GRANT CONNECT TO userid;
```

A user with no password cannot connect to the database. This is useful if you are creating a group and do not want anyone to connect to the database using the group user ID.

The `verify_password_function` option can be used to specify a function to implement password rules (for example, passwords must include at least one digit). If a password verification function is used, you cannot specify more than one user ID and password in the GRANT CONNECT statement.

Remarks

If you use this statement in a procedure, do not specify the password as a string literal because the definition of the procedure is visible in the SYSPROCEDURE system view. For security purposes, specify the password using a variable that is declared outside of the procedure definition.

Privileges

You must either be changing your own password using GRANT CONNECT, or have the MANAGE ANY USER privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following example creates a new database user named SQLTester, with password welcome.


```
GRANT CONNECT TO SQLTester  
IDENTIFIED BY welcome
```

Related Information

[CREATE USER statement](#)

Creates a database user or group.

[verify_password_function option](#)

Implements password rules.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

GRANT CONSOLIDATE statement [SQL Remote]

Identifies the database immediately above the current database in a SQL Remote hierarchy, that will receive messages from the current database.

Syntax

```
GRANT CONSOLIDATE  
TO userid  
TYPE message-system, ...  
ADDRESS address-string, ...  
[ SEND { EVERY | AT } hh:mm:ss ]
```

message-system :
FILE | FTP | SMTP

address : *string*

Parameters

- userid*** The user ID for the user to be granted the privileges.
- message-system*** One of the message systems supported by SQL Remote.
- address*** The address for the specified message system.

Remarks

In a SQL Remote installation, the database immediately above the current database in a SQL Remote hierarchy must be granted CONSOLIDATE privilege. GRANT CONSOLIDATE is issued at a remote database to identify its consolidated database. Each database can have only one user ID with CONSOLIDATE privileges: you cannot have a database that is a remote database for more than one consolidated database.

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 address-name must be a valid address for the message-system, enclosed in single quotes. There can be only one consolidated user per remote database.

For the FILE message type, the address is a subdirectory of the directory pointed to by the SQLREMOTE environment variable.

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 executed.

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 privileges without a SEND EVERY or SEND AT clause, the Message Agent processes messages, and then stops. To run the Message Agent continuously, you must ensure that every user with REMOTE privilege has either a SEND AT or SEND EVERY frequency

specified.

It is anticipated that at many remote databases, the Message Agent is run periodically, and that the consolidated database will have no SEND clause specified.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statement grants consolidated status to the Sam_Singer user ID:

```
GRANT CONSOLIDATE TO Sam_Singer  
TYPE SMTP  
ADDRESS 'Singer, Samuel';
```

Related Information

[SQL Remote message systems](#)

In SQL Remote replication, a message system is a protocol for exchanging messages between the consolidated database and a remote database. SQL Remote exchanges data among databases using one or more underlying message systems.

[Granting CONSOLIDATE privilege \(SQL Central\)](#)

Grant CONSOLIDATE privilege to a user.

[GRANT PUBLISH statement \[SQL Remote\]](#)

Grants publisher rights to a user ID. You must have the SYS_REPLICATION_ADMIN_ROLE system role to grant publisher rights.

[GRANT REMOTE statement \[SQL Remote\]](#)

Identifies a database immediately below the current database in a SQL Remote hierarchy, that will receive messages from the current database. These are called remote users.

[REVOKE CONSOLIDATE statement \[SQL Remote\]](#)

Stops a consolidated database from receiving SQL Remote messages from this database.

[ALTER REMOTE MESSAGE TYPE statement \[SQL Remote\]](#)

Changes the publisher's message system, or the publisher's address for a given message system, for a message type that has been created.

[CREATE REMOTE \[MESSAGE\] TYPE statement \[SQL Remote\]](#)

Identifies a message link and return address for outgoing messages from a database.

[CREATE SUBSCRIPTION statement \[SQL Remote\]](#)

Creates a subscription for a user to a publication.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

GRANT CREATE statement

Allows a user to create database objects in the specified dbspace.

Syntax

```
GRANT CREATE ON dbspace-name  
TO userid, ...
```

Remarks

When you initialize a database, it contains one database file. This first database file is called the main file. All database objects and all data are placed, by default, in the main file. A dbspace is an additional database file that creates more space for data. A database can be held in up to 13 separate files (the main file and 12 dbspaces). Each table, together with its indexes, must be contained in a single database file. The SQL command CREATE DBSPACE adds a new file to the database.

Privileges

You must have the MANAGE ANY USER privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Related Information

[Database file types](#)

Each database has several files associated with it.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

GRANT EXECUTE statement

Grants a user privilege to execute a procedure or user-defined function.

Syntax

```
GRANT EXECUTE ON [ owner. ] { procedure-name | user-defined-function }  
TO userid, ...
```

Remarks

None.

Privileges

You must own the procedure, or have the `MANAGE ANY OBJECT PRIVILEGE` system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Core Feature.

Example

The following example allows user `SQLTester` to execute the `Calculate_Report` procedure.

```
GRANT EXECUTE ON Calculate_Report  
TO SQLTester;
```

The following SQL statement grants `M_Haneef` the privilege required to execute a procedure named `my_procedure`.

```
GRANT EXECUTE  
ON my_procedure  
TO M_Haneef;
```

Related Information

[System procedures](#)

There are hundreds of system in the software, many of which are for internal use only. The documentation explains the system procedures that are for external use.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

GRANT INTEGRATED LOGIN statement

Grants a user privilege to execute a procedure or user-defined function.

Syntax

```
GRANT INTEGRATED LOGIN TO user-profile-name, ...  
[ AS USER userid ]
```

Remarks

The GRANT INTEGRATED LOGIN statement creates an explicit integrated login mapping between one or more Windows user or group profiles and an existing database user ID, allowing users who successfully log in to their local computer to connect to a database without having to provide a user ID or password. The *user-profile-name* can be of the form *domain\user-name*. The database user ID the integrated login is mapped to must have a password.

Privileges

You must have the MANAGE ANY USER system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Related Information

[Windows integrated login](#)

The Windows **integrated login** feature allows you to maintain a single user ID and password for operating system and network logins, and database connections.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

GRANT KERBEROS LOGIN statement

Creates a Kerberos authenticated login mapping from one or more Kerberos principals to an existing database user ID.

Syntax

```
GRANT KERBEROS LOGIN TO client-Kerberos-principal, ...  
AS USER userid
```

Remarks

The GRANT KERBEROS LOGIN statement creates a Kerberos authenticated login mapping from one or more Kerberos principals to an existing database user ID. This login mapping allows users who have successfully logged in to Kerberos (users who have a valid Kerberos ticket-granting ticket) to connect to a database without having to provide a user ID or password.

To use the GRANT KERBEROS LOGIN statement:

- You must already have Kerberos configured to use SQL Anywhere.
- You must already have your SQL Anywhere database configured to use Kerberos.
- The database user and the Kerberos principal must already exist.
- The database user ID the Kerberos login is mapped to must have a password.
- The *client-Kerberos-principal* must have the format *user/instance@REALM*, where */instance* is optional. The full principal, including the realm, must be specified, and principals that differ only in the instance or realm are treated as different.
- Principals are case sensitive and must be specified in the correct case. Mappings for multiple principals that differ only in case are not supported (for example, you cannot have mappings for both jjordan@MYREALM.COM and JJordan@MYREALM.COM).
- If no explicit mapping is made for a Kerberos principal, and the Guest database user ID exists and has a password, then the Kerberos principal connects using the Guest database user ID (the same Guest database user ID as for integrated logins).

Privileges

You must have the MANAGE ANY USER privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following SQL statement grants database login privilege to a fictitious Kerberos user pchin.

```
GRANT KERBEROS LOGIN TO "pchin@MYREALM.COM"  
AS USER "kerberos-user";
```

Related Information

[Kerberos user authentication](#)

The Kerberos login feature allows you to maintain a single user ID and password for database connections, operating system, and network logins.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

GRANT PUBLISH statement [SQL Remote]

Grants publisher rights to a user ID. You must have the SYS_REPLICATION_ADMIN_ROLE system role to grant publisher rights.

Syntax

GRANT PUBLISH TO *userid*

Remarks

Each database in a SQL Remote installation is identified in outgoing messages by a user ID, called the publisher. The GRANT PUBLISH statement sets the publisher user ID associated with these outgoing messages. To change publishers, you must revoke publisher rights from the current publisher (REVOKE PUBLISH), and then grant them to the new publisher.

The database publisher can be determined by querying the special value CURRENT PUBLISHER as follows:

```
SELECT CURRENT PUBLISHER;
```

If there is no publisher, the value of CURRENT PUBLISHER is NULL.

The CURRENT PUBLISHER special value 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 configuration helps prevent primary key conflicts due to updates at more than one site.

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.

Executing this statement changes the value of the PUBLIC.db_publisher database option.

Privileges

You must have the SET ANY SYSTEM OPTION system privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statement sets the database publisher to user ID JohnS.

```
GRANT PUBLISH TO JohnS;
```

The following statements revoke database publisher from JohnS and grant it to IrisM.

```
REVOKE PUBLISH FROM JohnS;  
GRANT PUBLISH TO IrisM;
```

Related Information

[Creating a publisher \(SQL Central\)](#)

Create users and grant them PUBLISH privilege.

[GRANT ROLE statement](#)

Grant roles to users and roles.

[db_publisher option](#)

Stores the user ID of the database publisher, if any.

[GRANT CONSOLIDATE statement \[SQL Remote\]](#)

Identifies the database immediately above the current database in a SQL Remote hierarchy, that will receive messages from the current database.

[REVOKE PUBLISH statement \[SQL Remote\]](#)

Terminates the identification of the named user ID as the current publisher. You must have the SYS_REPLICATION_ADMIN_ROLE system role to revoke publisher rights.

[CREATE SUBSCRIPTION statement \[SQL Remote\]](#)

Creates a subscription for a user to a publication.

[CURRENT PUBLISHER special value](#)

CURRENT PUBLISHER returns a string that contains the publisher user ID of the database for SQL Remote replications.

[CREATE REMOTE \[MESSAGE\] TYPE statement \[SQL Remote\]](#)

Identifies a message link and return address for outgoing messages from a database.

[GRANT REMOTE statement \[SQL Remote\]](#)

Identifies a database immediately below the current database in a SQL Remote hierarchy, that will receive messages from the current database. These are called remote users.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

GRANT REMOTE statement [SQL Remote]

Identifies a database immediately below the current database in a SQL Remote hierarchy, that will receive messages from the current database. These are called remote users.

Syntax

```
GRANT REMOTE TO userid, ...  
TYPE message-system, ...  
ADDRESS address-string, ...  
[ SEND { EVERY | AT } send-time ]
```

Parameters

userid The user ID for the user to be granted the privilege

message-system One of the message systems supported by SQL Remote. It must be one of the following values:

- FILE
- FTP
- SMTP

address-string A string containing a valid address for the specified message system.

send-time A string containing a time specification in the form *hh:mm:ss*.

Remarks

In a SQL Remote installation, each database receiving messages from the current database must be granted REMOTE privilege.

The single exception is the database immediately above the current database in a SQL Remote hierarchy, which must be granted CONSOLIDATE privilege.

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 address-name must be a valid address for the message-system, enclosed in single quotes.

For the FILE message type, the address is a subdirectory of the directory pointed to by the SQLREMOTE environment variable.

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.

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 privilege without a SEND EVERY or SEND AT clause, the Message Agent processes messages, and then stops. To run the Message Agent continuously, you must ensure that every user with REMOTE privilege has either a SEND AT or SEND EVERY frequency specified.

It is anticipated that at many consolidated databases, the Message Agent is 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.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Not in the standard.

Example

The following statement grants remote privilege to user Sam_Singer, using an SMTP email system, sending messages to the address Singer, Samuel once every two hours:

```
GRANT REMOTE TO Sam_Singer
TYPE SMTP
ADDRESS 'Singer, Samuel'
SEND EVERY '02:00';
```

Related Information

[REMOTE privilege](#)

Granting REMOTE privilege is also referred to as adding a remote user to the database. Publishers of databases directly below the current database in a SQL Remote hierarchy are granted REMOTE privilege by the current database.

[Subscriptions](#)

You subscribe a user to a publication by creating a **subscription**. Each database that shares information in a publication must have a subscription to the publication.

[Send frequency](#)

To run the SQL Remote Message Agent (dbremote) in continuous mode, for example on the consolidated database, ensure that every REMOTE user has a specified send frequency.

[SQL Remote message systems](#)

In SQL Remote replication, a message system is a protocol for exchanging messages between the consolidated database and a remote database. SQL Remote exchanges data among databases using one or more underlying message systems.

[Granting REMOTE privilege \(SQL Central\)](#)

Add a remote user or change an existing user to a remote user.

[REVOKE REMOTE statement \[SQL Remote\]](#)

Stops a user from being able to receive SQL Remote messages from this database.

[GRANT PUBLISH statement \[SQL Remote\]](#)

Grants publisher rights to a user ID. You must have the SYS_REPLICATION_ADMIN_ROLE system role to grant publisher rights.

[GRANT CONSOLIDATE statement \[SQL Remote\]](#)

Identifies the database immediately above the current database in a SQL Remote hierarchy, that will receive messages from the current database.

[CREATE SUBSCRIPTION statement \[SQL Remote\]](#)

Creates a subscription for a user to a publication.

[CREATE REMOTE \[MESSAGE\] TYPE statement \[SQL Remote\]](#)

Identifies a message link and return address for outgoing messages from a database.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

GRANT USAGE ON SEQUENCE statement

Grants privilege to use a specified sequence.

Syntax

```
GRANT USAGE ON SEQUENCE sequence-name  
TO userid, ...
```

Remarks

This privilege allows the user to select the current or next value from the specified sequence.

Privileges

You must have the MANAGE ANY OBJECT PRIVILEGE privilege.

Side effects

Automatic commit.

Standards

- **ANSI/ISO SQL Standard** Part of optional ANSI/ISO SQL Language Feature T176.

Example

A ticketing application creates a sequence to ensure that each ticket has an unique number.

```
CREATE SEQUENCE TicketNumber  
  MINVALUE 1000  
  MAXVALUE 100000;
```

A user, TicketAgent, using this application to generate tickets must be granted the usage rights to the sequence so that they can select values from the sequence.

```
GRANT USAGE ON SEQUENCE TicketNumber to TicketAgent;
```

Now, TicketAgent can select values from the TicketNumber sequence and use them to create new tickets. For example:

```
INSERT INTO NewTicket  
  SELECT TicketNumber.nextval, 'Concert Ticket', ...;
```

Related Information

[Use of a sequence to generate unique values](#)

You can use a **sequence** to generate values that are unique across multiple tables or that are different from a set of natural numbers.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
 » [Alphabetical list of SQL statements](#)

GROUP BY clause

Groups columns, alias names, and functions as part of the SELECT statement.

Syntax

GROUP BY

```
| group-by-term, ...
| simple-group-by-term, ... WITH ROLLUP
| simple-group-by-term, ... WITH CUBE
| GROUPING SETS ( group-by-term, ... )
```

group-by-term :

simple-group-by-term

```
| ( simple-group-by-term, ... )
```

```
| ROLLUP ( simple-group-by-term, ... )
```

```
| CUBE ( simple-group-by-term, ... )
```

simple-group-by-term :

expression

```
| ( expression )
```

```
| ( )
```

Parameters

GROUPING SETS clause The GROUPING SETS clause allows you to perform aggregate operations on multiple groupings from a single query specification. Each set specified in a GROUPING SET clause is equivalent to a GROUP BY clause.

For example, the following two queries are equivalent:

A grouping expression may be reflected in the result set as a NULL value, depending on the grouping in which the result row belongs. This may cause confusion over whether the NULL is the result of another grouping, or whether the NULL is the result of an actual NULL value in the underlying data. To distinguish between NULL values present in the input data and NULL values inserted by the grouping operator, use the GROUPING function.

Specifying an empty set of parentheses () in the GROUPING SETS clause returns a single row containing the overall aggregate.

ROLLUP clause The ROLLUP clause is similar to the GROUPING SETS clause in that it can be used to specify multiple grouping specifications within a single query specification. A ROLLUP clause of *s* generates +1 grouping sets, formed by starting with the empty parentheses, and then appending successive *s* from left to right.

For example, the following two statements are equivalent:

You can use a ROLLUP clause within a GROUPING SETS clause.

CUBE clause The CUBE clause is similar to the ROLLUP and GROUPING SETS clauses in that

it can be used to specify multiple grouping specifications within a single query specification. The CUBE clause is used to represent all possible combinations that can be made from the expressions listed in the CUBE clause.

For example, the following two statements are equivalent:

You can use a CUBE clause within a GROUPING SETS clause.

WITH ROLLUP clause This is an alternative syntax to the ROLLUP clause, and is provided for Transact-SQL compatibility.

WITH CUBE clause This is an alternate syntax to the CUBE clause, and is provided for Transact-SQL compatibility.

Remarks

When using the GROUP BY clause, you can group by expressions (with some limitations), columns, alias names, or functions. The result of the query contains one row for each distinct value (or set of values) of each grouping set.

The empty GROUP BY list, (), signifies the treatment of the entire input as a single group. For example, the following two statements are equivalent:

```
SELECT COUNT(), SUM(Salary) FROM GROUP0.Employees;
```

```
SELECT COUNT(), SUM(Salary) FROM GROUP0.Employees GROUP BY ();
```

Privileges

None.

Standards

- **ANSI/ISO SQL Standard** Core Feature. GROUPING SETS, GROUP BY (), ROLLUP, and CUBE constitute portions of optional ANSI/ISO SQL Language Feature T431. The software does not support optional ANSI/ISO SQL Language Feature T432, "Nested and concatenated GROUPING SETS".

Extensions to the GROUP BY clause that are not in the standard include:

Example

The following example returns a result set showing the total number of orders, and then provides subtotals for the number of orders in each year (2000 and 2001).

```
SELECT year ( OrderDate ) Year, Quarter ( OrderDate ) Quarter, count(*) Orders
FROM GROUP0.SalesOrders
GROUP BY ROLLUP ( Year, Quarter )
ORDER BY Year, Quarter;
```

Like the preceding ROLLUP operation example, the following CUBE query example returns a result set showing the total number of orders and provides subtotals for the number of orders in each year (2000 and 2001). Unlike ROLLUP, this query also gives subtotals for the number of orders in

each quarter (1, 2, 3, and 4).

```
SELECT year (OrderDate) Year, Quarter ( OrderDate ) Quarter, count(*) Orders
FROM GROUP0.SalesOrders
GROUP BY CUBE ( Year, Quarter )
ORDER BY Year, Quarter;
```

The following example returns a result set that gives subtotals for the number of orders in the years 2000 and 2001. The GROUPING SETS operation lets you select the columns to be subtotaled instead of returning all combinations of subtotals like the CUBE operation.

```
SELECT year (OrderDate) Year, Quarter ( OrderDate ) Quarter, count(*) Orders
FROM GROUP0.SalesOrders
GROUP BY GROUPING SETS ( ( Year, Quarter ), ( Year ) )
ORDER BY Year, Quarter;
```

Related Information

[Summarizing, grouping, and sorting query results](#)

Several procedures and statement clauses are supported to allow you to group and sort query results.

[GROUP BY clause extensions](#)

The standard GROUP BY clause of a SELECT statement allows you to group rows in the result set according the grouping expressions you supply.

[GROUP BY and the SQL/2008 standard](#)

The SQL/2008 standard is considerably more restrictive in its syntax than SQL Anywhere.

[GROUP BY GROUPING SETS](#)

The GROUPING SETS clause allows you to group your results multiple ways, without having to use multiple SELECT statements to do so.

[The CUBE clause](#)

A **data cube** is an n -dimensional summarization of the input using every possible combination of GROUP BY expressions, using the CUBE clause.

[The ROLLUP clause](#)

You can specify a hierarchy of grouping attributes using the ROLLUP clause.

[SELECT statement](#)

Retrieves information from the database.

[GROUPING function \[Aggregate\]](#)

Identifies whether a column in a GROUP BY operation result set is NULL because it is part of a subtotal row, or NULL because of the underlying data.

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

HELP statement [Interactive SQL]

Provides help in the Interactive SQL environment.

Syntax

HELP ['*topic*']

Remarks

The HELP statement is used to access SQL Anywhere documentation.

The *topic* for help can be optionally specified. You must enclose *topic* in single quotes. In some help formats, the topic cannot be specified; in this case, a link to a general help page for Interactive SQL appears.

You can specify the following *topic* values:

- SQL Anywhere error codes
- SQL statement keywords (such as INSERT, UPDATE, SELECT, CREATE DATABASE)

Privileges

None.

Side effects

None.

Standards

- **ANSI/ISO SQL Standard**

Related Information

[Interactive SQL](#)

Interactive SQL is a tool that lets you execute SQL statements, run SQL script files, as well as view and compare plans.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

IF statement

Controls conditional execution of SQL statements.

Syntax

```
IF search-condition THEN statement-list  
[ ELSEIF { search-condition | operation-type } THEN statement-list ] ...  
[ ELSE statement-list ]  
{ END IF | ENDIF }
```

Remarks

The IF statement is a control statement that allows you to conditionally execute the first list of SQL statements whose *search-condition* evaluates to TRUE. If no *search-condition* evaluates to TRUE, and an ELSE clause exists, the *statement-list* in the ELSE clause is executed.

Execution resumes at the first statement after the END IF.

Note Do not confuse the syntax of the IF statement with that of the IF expression.

Privileges

None.

Side effects

None.

Standards

-

Example

The following procedure illustrates the use of the IF statement:

```
CREATE PROCEDURE TopCustomer2 (OUT TopCompany CHAR(35), OUT TopValue INT)
BEGIN
  DECLARE err_notfound EXCEPTION
  FOR SQLSTATE '02000';
  DECLARE curThisCust CURSOR FOR
  SELECT CompanyName, CAST(      sum(SalesOrderItems.Quantity *
  Products.UnitPrice) AS INTEGER) VALUE
  FROM Customers
  LEFT OUTER JOIN SalesOrders
  LEFT OUTER JOIN SalesOrderItems
  LEFT OUTER JOIN Products
  GROUP BY CompanyName;
  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;
```

Related Information

[Stored procedures, triggers, batches, and user-defined functions](#)

Procedures and triggers store procedural SQL statements in a database.

[BEGIN statement](#)

Specifies a compound statement.

[IF expressions](#)

An IF expression tests whether a condition is TRUE, FALSE, or UNKNOWN.

[Search conditions](#)

A search condition is the criteria specified for a WHERE clause, a HAVING clause, a CHECK clause, an ON phrase in a join, or an IF expression. A search condition is also called a predicate.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

IF statement [T-SQL]

Controls conditional execution of a SQL statement, as an alternative to the Watcom SQL IF statement.

Syntax

```
[ [ ] ]
```

Remarks

The Transact-SQL IF conditional and the ELSE conditional each control the execution of only a single SQL statement or compound statement (between the keywords BEGIN and END).

In comparison to the Watcom SQL IF statement, there is no THEN in the Transact-SQL IF statement. The Transact-SQL version also has no ELSEIF or END IF keywords.

Privileges

None.

Side effects

None.

Standards

Example

The following example illustrates the use of the Transact-SQL IF statement:

```
IF (SELECT max(ID) FROM sysobjects) < 100
    RETURN
ELSE
    BEGIN
        PRINT 'These are the user-created objects'
        SELECT name, type, ID
        FROM sysobjects
        WHERE ID < 100
    END
```

The following two statement blocks illustrate Transact-SQL and Watcom SQL compatibility:

```
/* Transact-SQL IF statement */
IF @v1 = 0
    PRINT '0'
ELSE IF @v1 = 1
    PRINT '1'
ELSE
    PRINT 'other'
/* Watcom SQL IF statement */
IF v1 = 0 THEN
    PRINT '0'
ELSEIF v1 = 1 THEN
    PRINT '1'
ELSE
    PRINT 'other'
END IF
```

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

INCLUDE statement [ESQL]

Includes a file into a source program to be scanned by the SQL preprocessor.

Syntax

Remarks

Privileges

Side effects

Standards

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

INPUT statement [Interactive SQL]

Imports data into a database table from an external file, from the keyboard, from an ODBC data source, or from a shapefile.

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

INSERT statement

May 29, 2015 | [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

[SQL Anywhere 17](#) » [SQL Anywhere Server - SQL Reference](#) » [SQL statements](#)
» [Alphabetical list of SQL statements](#)

| [Terms of Use](#) | [Copyright](#) | [Trademark](#) | [Privacy](#) | [Legal Disclosure](#)

| | | | |

