CHAPTER 17

Command Reference for Adaptive Server Anywhere

About this chapter

This chapter describes the SQL statements used for executing SQL Remote commands, and the system tables, used for storing information about the SQL Remote installation and its state.

Contents

ALTER PUBLICATION statement

Function

To alter the definition of a SQL Remote publication.

Syntax

ALTER PUBLICATION [*owner*.]*publication-name*
  ADD TABLE *article-description*
... | **MODIFY TABLE** *article-description*
  | { **DELETE** | **DROP** } **TABLE** [*owner*.]*table-name*
  | **RENAME** *publication-name*

Parameters

article-description:
 table-name [(*column-name*, ...)]
 ... [**WHERE** *search-condition*]
 ... [**SUBSCRIBE BY** *expression*]

Usage

Anywhere. This statement is applicable only to SQL Remote.

Permissions

Must have DBA authority, or be owner of the publication. Requires exclusive access to all tables referred to in the statement.

Side effects

Automatic commit.

See also

"CREATE PUBLICATION statement" on page 356
"DROP PUBLICATION statement" on page 364

Description

The ALTER PUBLICATION statement alters a SQL Remote publication in the database. 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 specification of the modified article must be entered.

Example

The following statement adds the **customer** table to the **pub_contact** publication.

```
ALTER PUBLICATION pub_contact (
   ADD TABLE customer
)
```

ALTER REMOTE MESSAGE TYPE statement

Function

To change 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-string*

Parameters

Parameter	Description
message-system	One of the message systems supported by SQL Remote. It must be one of the following values: ♦ FILE ♦ FTP ♦ MAPI ♦ SMTP ♦ VIM
address-string	A string containing a valid address for the specified message system.

Permissions

Must have DBA authority.

Side effects

Automatic commit.

See also

"CREATE REMOTE MESSAGE TYPE statement" on page 358

Description

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 executing the GRANT CONSOLIDATE statement in the remote database.

The address is the publisher's address under the specified message system. If it is an e-mail system, the address string must be a valid e-mail address. If it is a file-sharing system, the address string is a subdirectory of the directory specified by the SQLREMOTE environment variable, or of the current directory if that is not set. You can override this setting on the GRANT CONSOLIDATE statement at the remote database.

For the FILE link, the ALTER REMOTE MESSAGE TYPE statement also causes the Message Agent to look for incoming messages in the address given for each message type.

Example

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

```
CREATE REMOTE MESSAGE TYPE file
```

```
ADDRESS 'new_addr'
```

CREATE PUBLICATION statement

Function To create a publication for replication with SQL Remote.

Syntax

CREATE PUBLICATION [*owner*.]*publication-name*
   ... (**TABLE** *article-description*,...)

Parameters

article-description:
   table-name [(*column-name*, ...)]
   ... [**WHERE** *search-condition*]
   ... [**SUBSCRIBE BY** *expression*]

Permissions Must have DBA authority. Requires exclusive access to all tables referred to in the statement.

Side effects Automatic commit.

See also

"ALTER PUBLICATION statement" on page 353
"DROP PUBLICATION statement" on page 364

Description The CREATE PUBLICATION statement creates a publication in the database. A publication can be created for another user by specifying an owner name.

In SQL Remote, publishing is a two-way operation, as data can be entered at both consolidated and remote databases. In a SQL Remote installation, any consolidated database and all remote databases must have the same publication defined. Running the extraction utility from a consolidated database automatically executes the correct CREATE PUBLICATION statement in the remote database.

Article Publications are built from articles. Each article is a table or part of a table. An article may be a vertical partition of a table (a subset of the table's columns), a horizontal partition (a subset of the table's rows) or a vertical and horizontal partition.

SUBSCRIBE BY clause 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.

WHERE clause The WHERE clause is a way of defining the subset of rows of a table to be included in an article. It is useful if the same subset if to be received by all subscribers to the publication.

You can combine WHERE and SUBSCRIBE BY clauses in an article definition.

Example

♦ The following statement creates a simple publication:

```
CREATE PUBLICATION pub_contact (
   TABLE contact
)
```

CREATE REMOTE MESSAGE TYPE statement

Function To identify a message-link and return address for outgoing messages from a database.

Syntax

CREATE REMOTE MESSAGE TYPE *message-system*
 ...**ADDRESS** *address-string*

Parameters

Parameter	Description
message-system	One of the message systems supported by SQL Remote. It must be one of the following values: ♦ FILE ♦ FTP ♦ MAPI ♦ SMTP ♦ VIM
address-string	A string containing a valid address for the specified message system.

Permissions Must have DBA authority.

Side effects Automatic commit.

See also

"GRANT PUBLISH statement" on page 369
"GRANT REMOTE statement" on page 370
"GRANT CONSOLIDATE statement" on page 367
"Using message types" on page 228

Description 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 e-mail system, the address string must be a valid e-mail address. If it is a file-sharing system, the address string is a subdirectory of the directory set in the SQLREMOTE environment variable, or of the current directory if that is not set. You can override this setting on the GRANT CONSOLIDATE statement at the remote database.

For the FILE link, the CREATE REMOTE MESSAGE TYPE statement also causes the Message Agent to look for incoming messages in the address given for each message type.

The initialization utility creates message types automatically, without an address. Unlike other CREATE statements, the CREATE REMOTE MESSAGE TYPE statement does not give an error if the type exists; instead it alters the type.

Example

- When remote databases are extracted using the extraction utility, the following statement sets all recipients of FILE messages to send messages back to the company subdirectory of the SQLREMOTE environment variable.

  The statement also instructs DBREMOTE to look in the *company* subdirectory for incoming messages.

  ```
  CREATE REMOTE MESSAGE TYPE file
  ADDRESS 'company'
  ```

CREATE SUBSCRIPTION statement

Function

To create a subscription for a user to a publication.

Syntax

CREATE SUBSCRIPTION
… **TO** *publication-name* [(*subscription-value*)]
… **FOR** *subscriber-id*

Parameters

Parameter	Description
publication-name	The name of the publication to which the user is being subscribed. This may 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.
subscriber-id	The user ID of the subscriber to the publication. This user must have been granted REMOTE permissions.

Permissions

Must have DBA authority.

Side effects

Automatic commit.

See also

"DROP SUBSCRIPTION statement" on page 366
"GRANT REMOTE statement" on page 370
"SYNCHRONIZE SUBSCRIPTION statement" on page 381
"START SUBSCRIPTION statement" on page 379

Description

In a SQL Remote installation, data is organized into **publications** for replication. In order to receive SQL Remote messages, a **subscription** must be created for a user ID with REMOTE permissions.

If a string is supplied in the subscription, it is matched against each SUBSCRIBE BY expression in the publication. The subscriber receives all rows for which the value of the expression is equal to the supplied string.

In SQL Remote, publications and subscriptions are two-way relationships. If you create a subscription for a remote user to a publication on a consolidated database, you should also create a subscription for the consolidated database on the remote database. The extraction utility carries this out automatically.

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
```

CREATE TRIGGER statement

Function

To create a new trigger in the database. One form of trigger is designed specifically for use by SQL Remote.

Syntax

```
CREATE TRIGGER trigger-name trigger-time trigger-event
      [, trigger-event,..]
      ... [ 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 ) ]
      ... [ IF UPDATE ( column-name ) THEN
      ... [ { AND | OR } UPDATE ( column-name ) ] ... ]
                  ... compound-statement
      ... [ ELSEIF UPDATE ( column-name ) THEN
      ... [ { AND | OR } UPDATE ( column-name ) ] ...
                 ...  compound-statement
      ...  END IF ] ]
```

Parameters

```
trigger-time:
      BEFORE | AFTER | RESOLVE

trigger-event:
      DELETE | INSERT | UPDATE | UPDATE OF column-list
```

Usage

Anywhere.

Permissions

Must have RESOURCE authority and have ALTER permissions on the table, or must have DBA authority. CREATE TRIGGER puts a table lock on the table and thus requires exclusive use of the table.

Side effects

Automatic commit.

See also

"UPDATE statement" on page 382

Description

The CREATE TRIGGER statement creates a trigger associated with a table in the database and stores the trigger in the database.

Triggers can be triggered by one or more of the following events:

♦ **INSERT** Invoked whenever a new row is inserted into the table associated with the trigger.

♦ **DELETE** Invoked whenever a row of the associated table is deleted.

♦ **UPDATE** Invoked whenever a row of the associated table is updated.

♦ **UPDATE OF column-list** Invoked whenever a row of the associated table is updated and a column in the *column-list* has been modified.

BEFORE UPDATE triggers fire any time an update occurs on a row, regardless of whether or not the new value differs from the old value. AFTER UPDATE triggers will fire only if the new value is different from the old value.

Row and statement-level triggers

The trigger is declared as either a row-level trigger, in which case it executes before or after each row is modified, or as a statement-level trigger, in which case it executes after the entire triggering statement is completed.

Row-level triggers can be defined to execute BEFORE or AFTER the insert, update, or delete. Statement-level triggers execute AFTER the statement. The RESOLVE trigger time is for use with SQL Remote; it fires before row-level UPDATE or UPDATE OF column-lists only.

To declare a trigger as a row-level trigger, use the FOR EACH ROW clause. To declare a trigger as a statement-level trigger, you can either use a FOR EACH STATEMENT clause or omit the FOR EACH clause. For clarity, it is recommended that you enter the FOR EACH STATEMENT clause if declaring a statement-level trigger.

Order of firing

Triggers of the same type (insert, update, or delete) that fire at the same time (before, after, or resolve) can use the ORDER clause to determine the order that the triggers are fired.

Referencing deleted and inserted values

The REFERENCING OLD and REFERENCING NEW clauses allow you to refer to the deleted and inserted rows. For the purposes of this clause, an UPDATE is treated as a delete followed by an insert.

The REFERENCING REMOTE clause is for use with SQL Remote. It allows you to refer to the values in the VERIFY clause of an UPDATE statement. It should be used only with RESOLVE UPDATE or RESOLVE UPDATE OF column-list triggers.

The meaning of REFERENCING OLD and REFERENCING NEW differs, depending on whether the trigger is a row-level or a statement-level trigger. For row-level triggers, the REFERENCING OLD clause allows you to refer to the values in a row prior to an update or delete, and the REFERENCING NEW clause allows you to refer to the inserted or updated values. The OLD and NEW rows can be referenced in BEFORE and AFTER triggers. The REFERENCING NEW clause allows you to modify the new row in a BEFORE trigger before the insert or update operation takes place.

For statement-level triggers, the REFERENCING OLD and REFERENCING NEW clauses refer to declared temporary tables holding the old and new values of the rows. The default names for these tables are **deleted** and **inserted**.

The WHEN clause causes the trigger to fire only for rows where the search-condition evaluates to true.

Updating values with the same value

BEFORE UPDATE triggers fire any time an UPDATE occurs on a row, whether or not the new value differs from the old value. AFTER UPDATE triggers fire only if the new value is different from the old value.

Example

- When a new department head is appointed, update the **manager_id** column for employees in that department.

```
CREATE TRIGGER
tr_manager BEFORE UPDATE OF dept_head_id ON
department
REFERENCING OLD AS old_dept
NEW AS new_dept
FOR EACH ROW
BEGIN
   UPDATE employee
   SET employee.manager_id=new_dept.dept_head_id
   WHERE employee.dept_id=old_dept.dept_id
END
```

DROP PUBLICATION statement

Function	To drop a SQL Remote publication.
Syntax	**DROP PUBLICATION** [*owner*.]*publication-name*
Usage	Anywhere. This statement is applicable only to SQL Remote.
Permissions	Must have DBA authority.
Side effects	Automatic commit. All subscriptions to the publication are dropped.
See also	"CREATE PUBLICATION statement" on page 356
Description	The DROP PUBLICATION statement drops an existing publication from the database. Publication definitions are held at both consolidated and remote databases in a SQL Remote installation.

Example

```
DROP PUBLICATION pub_contact
```

DROP REMOTE MESSAGE TYPE statement

Function

To delete a message type definition from a database.

Syntax

DROP REMOTE MESSAGE TYPE *message-system*

Parameters

Parameter	Description
message-system	One of the message systems supported by SQL Remote. It must be one of the following values: ♦ FILE ♦ FTP ♦ MAPI ♦ SMTP ♦ VIM

Permissions

Must have DBA authority. To be able to drop the type, there must be no user granted REMOTE or CONSOLIDATE permissions with this type.

Side effects

Automatic commit.

See also

"CREATE REMOTE MESSAGE TYPE statement" on page 358
"ALTER REMOTE MESSAGE TYPE statement" on page 354
"Using message types" on page 228.

Description

The statement removes a message type from a database.

Example

The following statement drops the FILE message type from a database.

```
DROP REMOTE MESSAGE TYPE file
```

DROP SUBSCRIPTION statement

Function

To drop a subscription for a user from a publication.

Syntax

DROP SUBSCRIPTION TO *publication-name* [(*subscription-value*)]
... **FOR** *subscriber-id*,...

Parameters

Parameter	Description
publication-name	The name of the publication to which the user is being subscribed. This may include the owner of the publication.
subscription-value	A string that is compared to the subscription expression of the publication. This value is required because a user may have more than one subscription to a publication.
subscriber-id	The user ID of the subscriber to the publication.

Permissions

Must have DBA authority.

Side effects

Automatic commit.

See also

"CREATE SUBSCRIPTION statement" on page 360

Description

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.

Example

The following statement drops a subscription for the user ID **SamS** to the publication **pub_contact**.

```
DROP SUBSCRIPTION TO pub_contact
FOR SamS
```

GRANT CONSOLIDATE statement

Function

To identify the database immediately above the current database in a SQL Remote hierarchy, who will receive messages from the current database.

Syntax

GRANT CONSOLIDATE
… **TO** *userid*, ...
… **TYPE** *message-system*, ...
… **ADDRESS** *address-string*, ...
… [**SEND** { **EVERY** | **AT** }*'hh:mm'*]

Parameters

Parameter	Description
userid	The user ID for the user to be granted the permission
message-system	One of the message systems supported by SQL Remote. It must be one of the following values: ♦ FILE ♦ FTP ♦ MAPI ♦ SMTP ♦ VIM
address-string	A string containing a valid address for the specified message system.

Permissions

Must have DBA authority.

Side effects

Automatic commit.

See also

"GRANT REMOTE statement" on page 370
"REVOKE CONSOLIDATE statement" on page 375
"GRANT PUBLISH statement" on page 369

Description

In a SQL Remote installation, the database immediately above the current database in a SQL Remote hierarchy must be granted CONSOLIDATE permissions. GRANT CONSOLIDATE is issued at a remote database to identify its consolidated database. Each database can have only one user ID with CONSOLIDATE permissions: you cannot have a database that is a remote database for more than one consolidated database.

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.

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 aready issued.

The optional SEND EVERY and SEND AT clauses specify a frequency at which messages are sent. The string contains a time that is a length of time between messages (for SEND EVERY) or a time of day at which messages are sent (for SEND AT). With SEND AT, messages are sent once per day.

If a user has been granted remote permissions without a SEND EVERY or SEND AT clause, the Message Agent processes messages, and then stops. In order to run the Message Agent continuously, you must ensure that every user with REMOTE permission has either a SEND AT or SEND EVERY frequency specified.

It is anticipated that at many remote databases, the Message Agent will be run periodically, and that the consolidated database will have no SEND clause specified.

Example

```
GRANT CONSOLIDATE TO con_db
TYPE mapi
ADDRESS 'Consolidated Database'
```

GRANT PUBLISH statement

Function

To identify the publisher of the current database.

Syntax

GRANT PUBLISH TO *userid*

Permissions

Must have DBA authority.

Side effects

Automatic commit.

See also

"GRANT REMOTE statement" on page 370
"GRANT CONSOLIDATE statement" on page 367
"REVOKE PUBLISH statement" on page 376
"CREATE PUBLICATION statement" on page 356
"CREATE SUBSCRIPTION statement" on page 360

Description

Each database in a SQL Remote installation is identified in outgoing messages by a user ID, called the **publisher**. The GRANT PUBLISH statement identifies the publisher user ID associated with these outgoing messages.

Only one user ID can have PUBLISH authority. The user ID with PUBLISH authority is identified by the special constant CURRENT PUBLISHER. The following query identifies the current publisher:

```
SELECT CURRENT PUBLISHER
```

If there is no publisher, the special constant is NULL.

The current publisher special constant can be used as a default setting for columns. It is often useful to have a CURRENT PUBLISHER column as part of the primary key for replicating tables, as this helps prevent primary key conflicts due to updates at more than one site.

In order to change the publisher, you must first drop the current publisher using the REVOKE PUBLISH statement, and then create a new publisher using the GRANT PUBLISH statement.

Example

```
GRANT PUBLISH TO publisher_ID
```

GRANT REMOTE statement

Function

To identify a database immediately below the current database in a SQL Remote hierarchy, who will receive messages from the current database. These are called remote users.

Syntax

GRANT REMOTE TO *userid*, ...
... **TYPE** *message-system*, ...
... **ADDRESS** *address-string*, ...
... [**SEND** { **EVERY** | **AT** } *send-time*]

Parameters

Parameter	Description
userid	The user ID for the user to be granted the permission
message-system	One of the message systems supported by SQL Remote. It must be one of the following values: ♦ FILE ♦ FTP ♦ MAPI ♦ SMTP ♦ VIM
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*.

Permissions

Must have DBA authority.

Side effects

Automatic commit.

See also

"GRANT CONSOLIDATE statement" on page 367
"REVOKE REMOTE statement" on page 377
"GRANT PUBLISH statement" on page 369
"Granting and revoking REMOTE and CONSOLIDATE permissions" on page 221,

Description

In a SQL Remote installation, each database receiving messages from the current database must be granted REMOTE permissions.

The single exception is the database immediately above the current database in a SQL Remote hierarchy, which must be granted CONSOLIDATE permissions.

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 permissions without a SEND EVERY or SEND AT clause, the Message Agent processes messages, and then stops. In order to run the Message Agent continuously, you must ensure that every user with REMOTE permission has either a SEND AT or SEND EVERY frequency specified.

It is anticipated that at many consolidated databases, the Message Agent will be run continuously, so that all remote databases would have a SEND clause specified. A typical setup may involve sending messages to laptop users daily (SEND AT) and to remote servers every hour or two (SEND EVERY). You should use as few different times as possible, for efficiency.

Example

♦ The following statement grants remote permissions to user **SamS**, using a MAPI e-mail system, sending messages to the address **Singer, Samuel** once every two hours:

```
GRANT REMOTE TO SamS
TYPE mapi
ADDRESS 'Singer, Samuel'
SEND EVERY '02:00'
```

GRANT REMOTE DBA statement

Function

To provide DBA privileges to a user ID, but only when connected from the Message Agent.

Syntax 1

GRANT REMOTE DBA
    TO *userid*,...
    IDENTIFIED BY *password*

Permissions

Must have DBA authority.

Side effects

Automatic commit.

See also

"The Message Agent and replication security" on page 257
"REVOKE REMOTE DBA statement" on page 378

Description

REMOTE DBA authority enables the Message Agent to have full access to the database in order to make any changes contained in the messages, while avoiding security problems associated with distributing DBA user IDs passwords.

REMOTE DBA has the following properties.

- No distinct permissions when not connected from the Message Agent. A user ID granted REMOTE DBA authority has no extra privileges on any connection apart from the Message Agent. Even if the user ID and password for a REMOTE DBA user is widely distributed, there is no security problem. As long as the user ID has no permissions beyond CONNECT granted on the database, no one can use this user ID to access data in the database.
- Full DBA permissions when connected from the Message Agent.

PASSTHROUGH statement

Function

To start or stop passthrough mode for SQL Remote administration. Forms 1 and 2 start passthrough mode, while form 3 stops passthrough mode.

Syntax 1

PASSTHROUGH [**ONLY**] **FOR** *userid*,...

Syntax 2

PASSTHROUGH [**ONLY**] **FOR SUBSCRIPTION**
... **TO** [(*owner*)].*publication-name* [(*constant*)]

Syntax 3

PASSTHROUGH STOP

Permissions

Must have DBA authority.

Side effects

None.

Description

In passthrough mode, any SQL statements are executed by the database server, and are also placed into the transaction log to be sent in messages to subscribers. If the ONLY keyword is used to start passthrough mode, the statements are not executed at the server; they are sent to recipients only. The recipients of the passthrough SQL statements are either a list of user IDs (form 1) or all subscribers to a given publication. Passthrough mode may be used to apply changes to a remote database from the consolidated database or send statements from a remote database to the consolidated database.

Syntax 2 sends statements to remote databases whose subscriptions are started, and does not send statements to remote databases whose subscriptions are created and not started.

Example

```
PASSTHROUGH FOR rem_db ;
...
( SQL statements to be executed at the remote database )
...
PASSTHROUGH STOP ;
```

REMOTE RESET statement

Function

For use in custom database extraction procedures. It starts all subscriptions for a remote user in a single transaction.

Syntax

REMOTE RESET *userid*

Permissions

Must have DBA authority.

Side effects

No automatic commit is done by this statement.

See also

"START SUBSCRIPTION statement" on page 379

Description

This command starts all subscriptions for a remote user in a single transaction. It sets the **log_sent** and **confirm_sent** values in **SYSREMOTEUSER** table to the current position in the transaction log. It also sets the created and started values in **SYSSUBSCRIPTION** to the current position in the transaction log for all subscriptions for this remote user. The statement does not do a commit. You must do an explicit commit after this call.

In order to write an extraction process that is safe on a live database, the data must be extracted at isolation level 3 in the same transaction as the subscriptions are started.

This statement is an alternative to start subscription. START SUBSCRIPTION has an implicit commit as a side effect, so that if a remote user has several subscriptions, it is impossible to start them all in one transaction using START SUBSCRIPTION.

Example

♦ The following statement resets the subscriptions for remote user SamS:

```
REMOTE RESET 'SamS'
```

REVOKE CONSOLIDATE statement

Function

To stop a consolidated database from receiving SQL Remote messages from this database.

Syntax

REVOKE CONSOLIDATE FROM *userid*,...

Permissions

Must have DBA authority.

Side effects

Automatic commit. Drops all subscriptions for the user.

See also

"GRANT CONSOLIDATE statement" on page 367

Description

CONSOLIDATE permissions must be granted at a remote database for the user ID representing the consolidated database. The REVOKE CONSOLIDATE statement removes the consolidated database user ID from the list of users receiving messages from the current database.

Example

♦ The following statement revokes consolidated status from the user ID **condb**:

```
REVOKE CONSOLIDATE FROM condb
```

REVOKE PUBLISH statement

Function

To terminate the identification of the named user ID as the CURRENT publisher.

Syntax

REVOKE PUBLISH FROM *userid*

Permissions

Must have DBA authority.

Side effects

Automatic commit.

See also

"GRANT PUBLISH statement" on page 369
"REVOKE REMOTE statement" on page 377
"CREATE PUBLICATION statement" on page 356
"CREATE SUBSCRIPTION statement" on page 360

Description

Each database in a SQL Remote installation is identified in outgoing messages by a **publisher** user ID. The current publisher user ID can be found using the CURRENT PUBLISHER special constant. The following query identifies the current publisher:

```
SELECT CURRENT PUBLISHER
```

The REVOKE PUBLISH statement ends the identification of the named user ID as the publisher.

You should not REVOKE PUBLISH from a database while the database has active SQL Remote publications or subscriptions.

Issuing a REVOKE PUBLISH statement at a database has several consequences for a SQL Remote installation:

- You will not be able to insert data into any tables with a CURRENT PUBLISHER column as part of the primary key. Any outgoing messages will not be identified with a publisher user ID, and so will not be accepted by recipient databases.

If you change the publisher user ID at any consolidated or remote database in a SQL Remote installation, you must ensure that the new publisher user ID is granted REMOTE permissions on all databases receiving messages from the database. This will generally require all subscriptions to be dropped and recreated.

Example

```
REVOKE PUBLISH FROM publisher_ID
```

REVOKE REMOTE statement

Function	To stop a user from being able to receive SQL Remote messages from this database.
Syntax	**REVOKE REMOTE FROM** *userid*,...
Permissions	Must have DBA authority.
Side effects	Automatic commit. Drops all subscriptions for the user.
Description	REMOTE permissions are required for a user ID to receive messages in a SQL Remote replication installation. The REVOKE REMOTE statement removes a user ID from the list of users receiving messages from the current database.

Example

```
REVOKE REMOTE FROM SamS
```

REVOKE REMOTE DBA statement

Function

To provide DBA privileges to a user ID, but only when connected from the Message Agent.

Syntax 1

REVOKE REMOTE DBA
    FROM *userid*,...

Permissions

Must have DBA authority.

Side effects

Automatic commit.

See also

"The Message Agent and replication security" on page 257
"GRANT REMOTE DBA statement" on page 372

Description

REMOTE DBA authority enables the Message Agent to have full access to the database in order to make any changes contained in the messages, while avoiding security problems associated with distributing DBA user IDs passwords.

This statement revokes REMOTE DBA authority from a user ID.

START SUBSCRIPTION statement

Function

To start a subscription for a user to a publication.

Syntax

START SUBSCRIPTION
… **TO** *publication-name* [(*subscription-value*)]
... **FOR** *subscriber-id*,...

Parameters

Parameter	Description
publication-name	The name of the publication to which the user is being subscribed. This may include the owner of the publication.
subscription-value	A string that is compared to the subscription expression of the publication. The value is required here because each subscriber may have more than one subscription to a publication.
subscriber-id	The user ID of the subscriber to the publication. This user must have a subscription to the publication.

Permissions

Must have DBA authority.

Side effects

Automatic commit.

See also

"CREATE SUBSCRIPTION statement" on page 360
"REMOTE RESET statement" on page 374
"SYNCHRONIZE SUBSCRIPTION statement" on page 381

Description

A SQL Remote subscription is said to be **started** when publication updates are being sent from the consolidated database to the remote database.

The START SUBSCRIPTION statement is one of a set of statements that manage subscriptions. The CREATE SUBSCRIPTION statement defines the data that the subscriber is to receive. The SYNCHRONIZE SUBSCRIPTION statement ensures that the consolidated and remote databases are consistent with each other. The START SUBSCRIPTION statement is required to start messages being sent to the subscriber.

Data at each end of the subscription must be consistent before a subscription is started. It is recommended that you use the database extraction utility to manage the creation, synchronization, and starting of subscriptions. If you use the database extraction utility, you do not need to execute an explicit START SUBSCRIPTION statement. Also, the Message Agent starts subscriptions once they are synchronized.

Example

The following statement starts the subscription of user **SamS** to the **pub_contact** publication.

```
START SUBSCRIPTION TO pub_contact
FOR SamS
```

STOP SUBSCRIPTION statement

Function

To stop a subscription for a user to a publication.

Syntax

STOP SUBSCRIPTION
… **TO** *publication-name* [(*subscription-value*)]
... **FOR** *subscriber-id*,...

Parameters

Parameter	Description
publication-name	The name of the publication to which the user is being subscribed. This may include the owner of the publication.
subscription-value	A string that is compared to the subscription expression of the publication. The value is required here because each subscriber may have more than one subscription to a publication.
subscriber-id	The user ID of the subscriber to the publication. This user must have a subscription to the publication.

Permissions

Must have DBA authority.

Side effects

Automatic commit.

See also

"CREATE SUBSCRIPTION statement" on page 360
"SYNCHRONIZE SUBSCRIPTION statement" on page 381

Description

A SQL Remote subscription is said to be **started** when publication updates are being sent from the consolidated database to the remote database.

The STOP SUBSCRIPTION statement prevents any further messages being sent to the subscriber. The START SUBSCRIPTION statement is required to restart messages being sent to the subscriber. However, you should ensure that the subscription is properly synchronized before restarting: that no messages have been missed.

Example

The following statement starts the subscription of user **SamS** to the **pub_contact** publication.

```
STOP SUBSCRIPTION TO pub_contact
FOR SamS
```

SYNCHRONIZE SUBSCRIPTION statement

Function

To synchronize a subscription for a user to a publication.

Syntax

SYNCHRONIZE SUBSCRIPTION
… **TO** *publication-name* [(*subscription-value*)]
... **FOR** *remote-user*,...

Parameters

Parameter	Description
publication-name	The name of the publication to which the user is being subscribed. This may include the owner of the publication.
subscription-value	A string that is compared to the subscription expression of the publication. The value is required here because each subscriber may have more than one subscription to a publication.
remote-user	The user ID of the subscriber to the publication. This user must have a subscription to the publication.

Permissions

Must have DBA authority.

Side effects

Automatic commit.

See also

"CREATE SUBSCRIPTION statement" on page 360
"START SUBSCRIPTION statement" on page 379

Description

A SQL Remote subscription is said to be **synchronized** when the data in the remote database is consistent with that in the consolidated database, so that publication updates sent from the consolidated database to the remote database will not result in conflicts and errors.

To synchronize a subscription, a copy of the data in the publication at the consolidated database is sent to the remote database. The SYNCHRONIZE SUBSCRIPTION statement does this through the message system. It is recommended that where possible you use the database extraction utility instead to synchronize subscriptions without using a message system.

Example

The following statement synchronizes the subscription of user **SamS** to the **pub_contact** publication.

```
SYNCHRONIZE SUBSCRIPTION TO pub_contact
FOR SamS
```

UPDATE statement

Function

To modify data in the database.

Syntax 1

UPDATE *table-list*
... **SET** *column-name* = *expression*, ...
... [**FROM** *table-list*]
... [**WHERE** *search-condition*]
... [**ORDER BY** *expression* [**ASC** | **DESC**] ,...]

Syntax 2

UPDATE *table-list*
... **SET** *column-name* = *expression*, ...
... [**VERIFY** (*column-name*, ...) **VALUES** (*expression*, ...)]
... [**WHERE** *search-condition*]
... [**ORDER BY** *expression* [**ASC** | **DESC**] ,...]

Syntax 3

UPDATE *table*
…**PUBLICATION** *publication*
...{ **SUBSCRIBE BY** *expression* |
OLD SUBSCRIBE BY *expression*
NEW SUBSCRIBE BY *expression*
}
...**WHERE** *search-condition*

expression: *value* | *subquery*

Usage

Syntax 1 can be used anywhere.

Syntax 2 and Syntax 3 are applicable only to SQL Remote.

Syntax 3 with no OLD and NEW SUBSCRIBE BY expressions must be used in a BEFORE trigger.

Syntax 3 with OLD and NEW SUBSCRIBE BY expressions can be used anywhere.

Permissions

Must have UPDATE permission for the columns being modified.

Side effects

None.

See also

"CREATE TRIGGER statement" on page 361

Description

The UPDATE statement is used to modify rows of one or more tables. Each named column is set to the value of the expression on the right hand side of the equal sign. There are no restrictions on the *expression*. Even *column-name* can be used in the expression—the old value will be used.

If no WHERE clause is specified, every row will be updated. If a WHERE clause is specified, then only those rows which satisfy the search condition will be updated.

Normally, the order that rows are updated doesn't matter. However, in conjunction with the NUMBER(*) function, an ordering can be useful to get increasing numbers added to the rows in some specified order. Also, if you wish to do something like add 1 to the primary key values of a table, it is necessary to do this in descending order by primary key, so that you do not get duplicate primary keys during the operation.

Views can be updated provided the SELECT statement defining the view does not contain a GROUP BY clause, an aggregate function, or involve a UNION operation.

Character strings inserted into tables are always stored in the case they are entered, regardless of whether the database is case sensitive or not. Thus a character data type column updated with a string **Value** is always held in the database with an upper-case V and the remainder of the letters lower case. SELECT statements return the string as **Value**. If the database is not case-sensitive, however, all comparisons make **Value** the same as **value**, **VALUE**, and so on. Further, if a single-column primary key already contains an entry **Value**, an INSERT of **value** is rejected, as it would make the primary key not unique.

Updates based on joins

The optional FROM clause allows tables to be updated based on joins. If the FROM clause is present, the WHERE clause qualifies the rows of the FROM clause. Data is updated only in the table list immediately following the UPDATE keyword.

If a FROM clause is used, it is important to qualify the table name that is being updated the same way in both parts of the statement. If a correlation name is used in one place, the same correlation name must be used in the other. Otherwise, an error is generated.

SQL Remote updates

Syntax 2 is intended for use with SQL Remote only, in single-row updates executed by the Message Agent. The VERIFY clause contains a set of values that are expected to be present in the row being updated. If the values do not match, any RESOLVE UPDATE triggers are fired before the UPDATE proceeds. The UPDATE does not fail if the VERIFY clause fails to match.

Syntax 3 is intended for use with SQL Remote only. If no OLD and NEW expressions are used, it must be used inside a BEFORE trigger so that it has access to the relevant values. The purpose is to provide a full list of subscribe by values any time the list changes. It is placed in SQL Remote triggers so that the database server can compute the current list of SUBSCRIBE BY values. Both lists are placed in the transaction log.

The Message Agent uses the two lists to make sure that the row moves to any remote database that did not have the row and now needs it. The Message Agent also removes the row from any remote database that has the row and no longer needs it. A remote database that has the row and still needs it is not be affected by the UPDATE statement.

Performance tip

Syntax 3 of the UPDATE statement allows the old SUBSCRIBE BY list and the new SUBSCRIBE BY list to be explicitly specified, which can make SQL Remote triggers more efficient. In the absence of these lists, the database server computes the old SUBSCRIBE BY list from the publication definition. Since the new SUBSCRIBE BY list is commonly only slightly different from the old SUBSCRIBE BY list, the work to compute the old list may be done twice. By specifying both the old and new lists, this extra work can be avoided.

The OLD and NEW SUBSCRIBE BY syntax is especially useful when many tables are being updated in the same trigger with the same subscribe by expressions. This can dramatically increase performance.

The SUBSCRIBE BY expression is either a value or a subquery.

Using UPDATE to maintain subscriptions

Syntax 3 of the UPDATE statement is used to implement a specific SQL Remote feature, and is to be used inside a BEFORE trigger.

For publications created using a subquery in a subscription expression, you must write a trigger containing syntax 3 of the UPDATE statement in order to ensure that the rows are kept in their proper subscriptions.

For a full description of this feature, see "Territory realignment in the Contact example" on page 128.

Syntax 3 of the UPDATE statement makes an entry in the transaction log, but does not change the database table.

Examples

- Transfer employee Philip Chin (employee 129) from the sales department to the marketing department.

```
UPDATE employee
SET dept_id = 400
WHERE emp_id = 129 ;
```