CHAPTER 18

Command Reference for Adaptive Server Enterprise

About this chapter

This chapter describes the SQL Remote stored procedures, used for executing SQL Remote commands.

Contents

sp_add_article procedure

Purpose

To add an article to a publication.

Syntax

sp_add_article *publication_name, table_name, where_expr, subscribe_by_expr*

Argument	Description
publication_name	The name of the publication to which the article is to be added.
table_name	The table containing the article.
where_expr	This optional argument must be a column name or NULL. The publication includes only rows for which the supplied column value is not NULL. The default value is NULL, in which case no rows are excluded from the publication..
subscribe_by_expr	The new subscription expression defining which rows are to be included in the publication for each subscription. The expression must be the name of a column in *table_name*. The default value is NULL.

See also

"sp_add_remote_table procedure" on page 389
"sp_create_publication procedure" on page 391
"sp_remove_article procedure" on page 425

Description

Running **sp_add_article** adds an article to a publication. The table must be marked for replication using the "sp_add_remote_table procedure" on page 389 before it can be added to a publication; failure to do so leads to an error.

Calling **sp_add_article** adds all the columns of the table to a publication. If you wish to include only some of the columns of the table in a publication you must first run **sp_add_article** and then call the "sp_add_article_col procedure" on page 388.

As with other data definition changes, in a production environment this procedure should only be run on a quiet SQL Remote installation.

For more information on the requirements for a quiet system, see "Making schema changes" on page 289.

Example

- The following statement adds the SalesRep table to a publication named SalesRepData:

```
sp_add_article 'SalesRepData', 'SalesRep'
```

sp_add_article_col procedure

Purpose

To add a column to an article in a publication.

Syntax

sp_add_article_col *publication_name, table_name, column_name*

Argument	Description
publication_name	The name of the publication to which the article is to be added.
table_name	The table containing the article.
column_name	The column to be added to the article in a publication

See also

"sp_add_article procedure" on page 387
"sp_remove_article procedure" on page 425

Description

Running **sp_add_article_col** adds a column to an article in a publication. The table must first be added to the publication using the "sp_add_article procedure" on page 387.

To add all the columns of a table to a publication you do not need to use sp_add_article_col; just call sp_add_article.

To add only some of the columns of a table to a publication you first call sp_add_article, and then call **sp_add_article_col** for each of the columns you wish to include in the publication.

As with other data definition changes, in a production environment this procedure should only be run on a quiet SQL Remote installation.

For more information on the requirements for a quiet system, see "Making schema changes" on page 289.

Example

- The following statements add the emp_id and emp_lname columns of the employee table to a publication named Personnel:

```
sp_add_article 'Personnel', employee'
sp_add_article_col 'Personnel', 'employee', 'emp_id'
sp_add_article_col 'Personnel', 'employee',
'emp_lname'
go
```

sp_add_remote_table procedure

Purpose

To mark a table for SQL Remote replication.

Syntax

sp_add_remote_table *table_name,*
[*resolve_procedure,*]
[*old_row_name,*]
[*remote_row_name*]

Argument	Description
table_name	The table to be marked for SQL Remote replication.
resolve_procedure	The name of a stored procedure that carries out actions when a conflict occurs.
old_row_name	The name of a table holding the values in the table when a conflict occurs.
remote_row_name	The name of a table holding the values at the remote database when a conflict-causing UPDATE statement was applied.

Authorization

You must be a system administrator to execute this procedure.

See also

"sp_modify_remote_table procedure" on page 401
"sp_remove_remote_table procedure" on page 427
"Managing conflicts" on page 182.

Description

Each table in a database must be marked for replication by using **sp_add_remote_table** before it can be included in any SQL Remote publications. After executing **sp_add_remote_table**, you can add the table to a publication using the "sp_add_article procedure" on page 387 and the "sp_add_article_col procedure" on page 388.

The **sp_add_remote_table** procedure calls **sp_setreplicate**, which flags the table for replication. This tells Adaptive Server Enterprise to put extended information into the transaction log. This information includes the entire before and after images of the row.

The first argument is the name of the table to be marked for replication.

The remaining three arguments are optional. They are object names required only for custom conflict resolution. If you are implementing custom conflict resolution, you must supply the names of two tables, and a stored procedure. The **sp_add_remote_table** procedure does not check for the existence of the conflict resolution arguments: you can create them either before or after marking the table for replication.

The two tables must have the same columns and data types as table *table_name*.

Examples

- The following statement marks the Customer table for replication, using default conflict resolution:

```
exec sp_add_remote_table Customer
```

- The following statement marks the Customer table for replication, using a stored procedure named Customer_Conflict to resolve conflicts. The old and remote rows are stored in tables named old_Customer and remote_Customer, respectively:

```
exec sp_add_remote_table Customer,
Customer_Conflict, old_Customer, remote_Customer
```

sp_create_publication procedure

Purpose

To create a publication.

Syntax

sp_create_publication *publication_name*

Argument	Description
publication_name	The name of the publication

See also

"sp_drop_publication procedure" on page 392

Description

Running **sp_create_publication** creates a publication, but one with no content. Once the publication is created, you must add articles to it using the "sp_add_remote_table procedure" on page 389 and the "sp_add_article procedure" on page 387.

Example

- The following statement creates a publication named **SalesRepData**:

```
sp_create_publication 'SalesRepData'
go
```

sp_drop_publication procedure

Purpose

To drop a publication from the database.

Syntax

sp_drop_publication *publication_name*

Argument	Description
publication_name	The name of the publication to be dropped

See also

"sp_create_publication procedure" on page 391

Description

Running **sp_drop_publication** drops a publication from the database. All articles that make up the publication, and subscriptions to the publication, are also dropped.

Example

- The following statement drops the publication named SalesRep:

```
sp_drop_publication 'SalesRep'
    go
```

sp_drop_remote_type procedure

Purpose

To drop a message type from the database.

Syntax

sp_drop_remote_type *type_name*

Argument	Description
type_name	The message type to drop. This must be a string containing one of the following: ♦ file ♦ ftp ♦ smtp ♦ mapi ♦ vim

See also

"sp_remote_type procedure" on page 424

Description

Drops the named message type from the database.

Example

♦ The following statement drops the MAPI message type from the database:

```
sp_drop_remote_type mapi
go
```

sp_drop_sql_remote procedure

Purpose

To drop the SQL Remote system objects from a database.

Syntax

sp_drop_sql_remote

See also

"sp_queue_drop procedure" on page 413

Description

Drops the SQL Remote system objects from the database, so that it can no longer function in a SQL Remote installation.

The sole SQL Remote object not removed is the **sp_drop_sql_remote** procedure itself (a procedure cannot drop itself from a database). To complete removal of SQL Remote requires that sp_drop_sql_remote be dropped explicitly after it is called.

The **sp_drop_sql_remote** procedure does not remove stable queue objects from the database. To remove the stable queue, use the "sp_queue_drop procedure" on page 413.

Example

- The following statements remove SQL Remote system objects from a database:

```
sp_drop_sql_remote_type
go
drop procedure sp_drop_sql_remote
go
```

sp_grant_consolidate procedure

Purpose

To identify a database immediately above the current database in a SQL Remote hierarchy, who will receive messages from the current database. This procedure applies only to Adaptive Server Enterprise databases acting as remote databases.

Syntax

sp_grant_consolidate *user_name, type_name, address*
[, *frequency*] [, *send_time*]

Argument	Description
user_name	The user ID who will be able to receive SQL Remote messages.
type_name	The message type to drop. This must be one of the following: ♦ file ♦ ftp ♦ smtp ♦ mapi ♦ vim
address	A string holding the address, according to the specified message type, to which the replication messages should be sent for this user.
frequency	A string containing one of the following: ♦ **SEND EVERY** Indicates that messages are sent at a frequency specified by *send_time*. ♦ **SEND AT** Indicates that messages are sent at a time of day specified by *send_time*.
send_time	A string containing a time specification with the following meaning: ♦ If *frequency* is SEND EVERY, specifies a length of time between messages. ♦ If *frequency* is SEND AT, specifies a time of day at which messages will be sent. If no frequency is specified, the Message Agent sends messages, and then stops.

See also

"sp_grant_remote procedure" on page 397
"sp_revoke_consolidate procedure" on page 428

Description

If the Adaptive Server Enterprise server is acting as a remote database in a SQL Remote installation, the single database above the current database must be granted consolidated permissions using the **sp_grant_consolidate** procedure.

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.

The **sp_grant_consolidate** procedure 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.

The optional *frequency* argument specifies a frequency at which messages are sent. The *send_time* argument 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 no *frequency* argument is supplied, the Message Agent processes messages, and then stops. In order to run the Message Agent continuously, you must ensure that every user with remote or consolidated permission has a frequency specified.

Example

- The following statement grants consolidated permissions to user **hq_user**, using a file sharing system, sending messages to the address **hq_dir**: No frequency arguments are specified, and the Message Agent will run in batch mode.

```
sp_grant_consolidate
   @user_name=hq_user,
   @address=hq_dir,
   @type_name=file
go
```

sp_grant_remote procedure

Purpose

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

sp_grant_remote *user_name, type_name, address*
[, *frequency*] [, *send_time*]

Argument	Description
user_name	The user ID who will be able to receive SQL Remote messages.
type_name	The message type to drop. This must be one of the following: ♦ file ♦ ftp ♦ smtp ♦ mapi ♦ vim
address	A string holding the address, according to the specified message type, to which the replication messages should be sent for this user.
frequency	A string containing one of the following: ♦ **SEND EVERY** Indicates that messages are sent at a frequency specified by *send_time*. ♦ **SEND AT** Indicates that messages are sent at a time of day specified by *send_time*.
send_time	An optional string containing a time specification with the following meaning: ♦ If *frequency* is SEND EVERY, specifies a length of time between messages. ♦ If *frequency* is SEND AT, specifies a time of day at which messages will be sent. If no frequency is specified, the Message Agent sends messages, and then stops.

See also

"sp_revoke_remote procedure" on page 429

Description

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

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.

The **sp_grant_remote** procedure 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.

The optional *frequency* argument specifies a frequency at which messages are sent. The *send_time* argument 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 no *frequency* argument is supplied, 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 a 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 *frequency* argument 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:

```
exec sp_grant_remote 'SamS',
   'mapi',
   'Singer, Samual',
   'SEND EVERY',
   '02:00'
go
```

sp_modify_article procedure

Purpose

To change the description of an article in a procedure.

Syntax

sp_modify_article
publication_name,
table_name,
[*where_expr*,]
[*subscribe_by_expr*]

Argument	Description
publication_name	The name of the publication for which the article is to be modified.
table_name	The table containing the article.
where_expr	This optional argument must be a column name or NULL. The publication includes only rows for which the supplied column name is not NULL. The default value is NULL, in which case no rows are excluded from the publication..
subscribe_by_expr	The new subscription expression defining which rows are to be included in the publication for each subscription. The default value is NULL.

See also

"sp_add_article procedure" on page 387
"sp_remove_article procedure" on page 425

Description

To change the description of an article in a publication. The WHERE expression and the subscription expression can both be changed.

As with other data definition changes, in a production environment this procedure should only be run on a quiet SQL Remote installation.

For more information on the requirements for a quiet system, see "Making schema changes" on page 289.

Examples

- The following statement changes an article in the **SalesRepData** publication that takes information from the **Customer** table, so that it has no subscription expression:

```
sp_modify_article SalesRepData, Customer
go
```

- The following statement changes an article in the **SalesRepData** publication that takes information from the **Customer** table, so that it has a subscription expression that is the **rep_key** column:

```
sp_modify_article SalesRepData,
```

```
    Customer,
    NULL,
    rep_key
go
```

sp_modify_remote_table procedure

Purpose

To change the resolution objects for a table marked for SQL Remote replication.

Syntax

sp_modify_remote_table *table_name,*
[*resolve_name,*]
[*old_row_name,*]
[*remote_row_name*]

Argument	Description
table_name	A table marked for SQL Remote replication.
resolve_procedure	The name of the new stored procedure for carrying out actions when a conflict occurs.
old_row_name	The name of the new table for holding the values in the table when a conflict occurs.
remote_row_name	The name of the new table for holding the values at the remote database when a conflict-causing UPDATE statement was applied.

See also

"sp_add_remote_table procedure" on page 389
"sp_remove_remote_table procedure" on page 427
"Managing conflicts" on page 182.

Description

Each table in a database must be marked for replication by using **sp_add_remote_table** before it can be included in any SQL Remote publications.

The **sp_modify_remote_table** allows you to change the way in which conflict resolution is carried out for update conflicts occurring on this table.

The arguments are, in addition to the table name, the object names required for custom conflict resolution. If you are implementing custom conflict resolution, you must supply the names of two tables, and a stored procedure. The **sp_modify_remote_table** procedure does not check for the existence of the conflict resolution arguments: you can create them either before or after marking the table for replication.

The two tables must have the same columns and data types as table *table_name*.

Example

- The following statement instructs SQL Remote to use the resolve_Cust procedure, the old_Cust table, and the remote_Cust table to resolve update conflicts on the **Customer** table:

```
sp_add_remote_table Customer,
   resolve_Cust,
```

```
    old_Cust,
    remote_Cust
go
```

sp_passthrough procedure

Purpose

To send a SQL statement in passthrough mode.

Syntax

sp_passthrough *statement*

Argument	Description
statement	A string containing a statement to be executed in passthrough mode.

See also

"sp_passthrough_piece procedure" on page 404
"sp_passthrough_stop procedure" on page 406
"sp_passthrough_subscription procedure" on page 407
"sp_passthrough_user procedure" on page 408

Description

To send passthrough operations. The recipients of the passthrough statement are determined by previous calls to **sp_passthrough_user** and **sp_passthrough_subscription**.

The string must be less than 255 characters long. For SQL statements longer than 255 characters, you should execute a sequence of calls to the **sp_passthrough_piece** procedures, and execute **sp_passthrough** for the final piece of the statement and to cause the replication to occur.

> ***Caution***
> *You should always test your passthrough operations on a test database with a remote database subscribed. You should never run untested passthrough scripts against a production database.*

Example

- The following statement sends a create table statement to the current recipients of passthrough statements.

```
exec sp_passthrough
   'CREATE TABLE simple (
      id integer NOT NULL,
      name char(50) )'
go
```

sp_passthrough_piece procedure

Purpose

To build a long SQL statement for passthrough.

Syntax

sp_passthrough_piece *string*

Argument	Description
string	A piece of a statement to be executed in passthrough mode.

See also

"sp_passthrough procedure" on page 403
"sp_passthrough_stop procedure" on page 406
"sp_passthrough_subscription procedure" on page 407
"sp_passthrough_user procedure" on page 408

Description

The "sp_passthrough procedure" on page 403 is used to send statements directly to a set of remote users. Statements that are longer than 255 characters have to be built up piece by piece.

To build and send a long SQL statement, call **sp_passthrough_piece** for all but the final piece of the statement, and then call **sp_passthrough** for the final piece. This completes and replicates the statement.

All pieces of a passthrough statement must be built within a single transation.

Example

- The following statements send a long passthrough statement to the current list of passthrough recipients:

```
begin transaction
go
exec sp_passthrough_piece 'CREATE TABLE
   DBA.employee
   (
   emp_id  integer NOT NULL,
   manager_id  integer NULL,
   emp_fname  char(20) NOT NULL,
   emp_lname  char(20) NOT NULL,'
go
exec sp_passthrough_piece '
   dept_id  integer NOT NULL,
   street  char(40) NOT NULL,
   city  char(20) NOT NULL,
   state  char(4) NOT NULL,
   zip_code  char(9) NOT NULL,
   phone  char(10) NULL,'
go
exec sp_passthrough_piece 'status  char(1) NULL,
   ss_number  char(11) NOT NULL,
   salary  numeric(20,3) NOT NULL,
```

```
   start_date  date NOT NULL,
   termination_date  date NULL,
   birth_date  date NULL,'
go
exec sp_passthrough '
   bene_health_ins  char(1) NULL,
   bene_life_ins  char(1) NULL,
   bene_day_care  char(1) NULL,
   sex  char(1) NULL,
   PRIMARY KEY (emp_id),
   )'
go
commit
go
```

sp_passthrough_stop procedure

Purpose

Resents passthrough mode

Syntax

sp_passthrough_stop

See also

"sp_passthrough procedure" on page 403
"sp_passthrough_subscription procedure" on page 407
"sp_passthrough_user procedure" on page 408

Description

The sp_passthrough_stop procedure resents the list of recipients of passthrough statements to be empty, and clears any statements that are currently being built.

Example

- The following statement resets the passthrough recipient list to be empty.

```
exec sp_passthrough_stop
go
```

sp_passthrough_subscription procedure

Purpose

Adds subscribers to a given publication to the recipient list for passthrough statements.

Syntax

sp_passthrough_subscription *publication_name*, *subscribe_by*

Argument	Description
publication_name	The name of the publication
subscribe_by	The subscription value for recipients to receive passthrough statements.

See also

"sp_passthrough procedure" on page 403
"sp_passthrough_piece procedure" on page 404
"sp_passthrough_stop procedure" on page 406
"sp_passthrough_user procedure" on page 408

Description

This is one of two ways that you can add to the list of recipients for passthrough statements, the other being to use the "sp_passthrough_user procedure" on page 408.

The users that are added to the recipient list by a call to the **sp_passthrough_subscription** procedure are all those users subscribing to the publication *publication_name* with a subscription value of *subscribe_by*.

The default setting for *subscribe_by* is NULL. In this case, all subscribers to the publication receive the passthrough statements.

Example

- The following statement adds to the list of passthrough recipients the subscriber or subscribers to the **SalesRepData** publication who use subscription values of 'rep1'.

```
Sp_passthrough_subscription SalesRepData, rep1
```

sp_passthrough_user procedure

Purpose

Adds a named user to the list of recipients for passthrough statements.

Syntax

sp_passthrough_user *user_name*

Argument	Description
user_name	The user to be added to the list of recipients.

See also

"sp_passthrough procedure" on page 403
"sp_passthrough_piece procedure" on page 404
"sp_passthrough_stop procedure" on page 406
"sp_passthrough_subscription procedure" on page 407

Description

This is one of two ways that you can add to the list of recipients for passthrough statements, the other being to use the "sp_passthrough_subscription procedure" on page 407.

The sp_passthrough_user procedure adds the named user to the list of recipients for passthrough statements. The list remains in force until reset using the "sp_passthrough_stop procedure" on page 406.

Example

- The following statement adds the user field_user to the list of recipients for passthrough statements:

```
sp_passthrough_user 'field_user'
go
```

sp_populate_sql_anywhere procedure

Purpose

To create a copy of the Adaptive Server Anywhere system tables in the TEMPDB. This procedure is used by the extraction utility *ssxtract*.

Syntax

sp_populate_sql_anywhere

Description

To create a set of Adaptive Server Anywhere system tables for a remote Adaptive Server Anywhere database, in TEMPDB. The information is used by the extraction utility to construct an Adaptive Server Anywhere database schema from the set of publications in the Adaptive Server Enterprise consolidated database.

This procedure is used by the *ssxtract* extraction utility. It should not be called directly.

sp_publisher procedure

Purpose

To set the publisher of the current database, or to remove the publisher..

Syntax

sp_publisher [*user_name*]

Argument	Description
user_name	The user ID to be identifies as the publisher for the database.

See also

"Managing SQL Remote permissions" on page 219.

Description

Each database in a SQL Remote installation is identified in outgoing messages by a user ID, called the **publisher**. The **sp_publisher** procedure sets the publisher user ID associated with these outgoing messages.

Each database can have at most one publisher; if a publisher already exists, **sp_publisher** changes the name of the publisher.

If no *user_name* argument is provided, the current publisher is removed, so that the database has no publisher. Only the permission to be the publisher is removed; the user ID is not removed from the database.

Examples

- The following statement identifies the user ID **joe** as the publisher of the current database:

```
sp_publisher joe
go
```

- The following statement sets the current database to have no publisher:

```
sp_publisher
go
```

sp_queue_clean procedure

Purpose

This procedure is used by the SQL Remote Message Agent, and should not be called directly.

Syntax

sp_queue_clean

Description

This procedure is used by the SQL Remote Message Agent, and should not be called directly. It removes from the stable queue any transactions that completed after the start of the oldest incomplete transaction the last time the log was scanned.

sp_queue_delete_old procedure

Purpose

This procedure is used by the SQL Remote Message Agent, and should not be called directly.

Syntax

sp_queue_delete_old

Description

This procedure is used by the SQL Remote Message Agent, and should not be called directly. It deletes from the stable queue any transactions that have been confirmed by all remote databases.

sp_queue_drop procedure

Purpose

To drop the stable queue objects from a database..

Syntax

sp_queue_drop

See also

"sp_drop_sql_remote procedure" on page 394

Description

Drops the stable queue system objects from the database, so that the database no longer supports a SQL Remote stable queue.

The sole stable queue object not removed is the **sp_queue_drop** procedure itself (a procedure cannot drop itself from a database). To complete removal of the stable queue requires that **sp_queue_drop** be dropped explicitly after it is called.

The **sp_queue_drop** procedure does not remove SQL Remote system objects from the database. To remove the SQL Remote system objects, use the "sp_drop_sql_remote procedure" on page 394.

Examples

- The following statements remove the stable queue objects from the database:

```
sp_queue_drop
go
drop procedure sp_queue_drop
go
```

sp_queue_dump_database procedure

Purpose

To facilitate recovery from media failure when the stable queue is in a separate database from the SQL Remote objects.

Syntax

sp_queue_dump_database

See also

"sp_queue_dump_transaction procedure" on page 415
"Stable queue recovery issues" on page 287

Description

Keeping the stable queue in a separate database complicates backup and recovery, as consistent versions of the two databases have to be recovered.

Normal recovery automatically restores the two databases to a consistent state, but recovery from media failure takes some care. When restoring database dumps, it is important to recover the stable queue to a consistent point. The **sp_queue_dump_database** procedure is provided to help with recovery from media failure. It is called whenever a **dump database** is scanned.

As provided, the procedure does not carry out any operations. You can modify this stored procedure to issue a **dump database** command in the stable store database.

sp_queue_dump_transaction procedure

Purpose

To facilitate recovery from media failure, when the stable queue is in a separate database from the SQL Remote objects.

Syntax

sp_queue_dump_transaction

See also

"sp_queue_dump_database procedure" on page 414
"Stable queue recovery issues" on page 287

Description

Keeping the stable queue in a separate database complicates backup and recovery, as consistent versions of the two databases have to be recovered.

Normal recovery automatically restores the two databases to a consistent state, but recovery from media failure takes some care. When restoring database dumps, it is important to recover the stable queue to a consistent point. The **sp_queue_dump_transaction** procedure is provided to help with recovery from media failure. It is called whenever a **dump transaction** is scanned.

As provided, the procedure does not carry out any operations. You can modify this stored procedure to issue a **dump transaction** command in the stable store database.

sp_queue_get_state procedure

Purpose

This procedure is used by the SQL Remote Message Agent, and should not be called directly.

Syntax

sp_queue_get_state

Description

This procedure is used by the SQL Remote Message Agent, and should not be called directly. It returns a description of the current state of the stable queue.

sp_queue_read procedure

Purpose

This procedure is used by the SQL Remote Message Agent, and should not be called directly.

Syntax

sp_queue_read *start_offset*, *stop_offset*

Description

This procedure reads transactions from the stable queue. It is exclusively for use by the Message Agent.

sp_queue_reset procedure

Purpose

To reset the server to a point where the stable queue is empty.

Syntax

sp_queue_reset

Description

This procedure is used by the SQL Remote Message Agent, and should not be called directly in a production environment. It deletes all rows from the stable queue **sr_transaction** table, and resets the **sr_queue_state** table, ready for a new SQL Remote setup.

In a development phase, this procedure can be useful to reset the server.

sp_queue_set_confirm procedure

Purpose

This procedure is used by the SQL Remote Message Agent, and should not be called directly.

Syntax

sp_queue_set_confirm *confirm_offset*

Description

This procedure is used by the SQL Remote Message Agent, and should not be called directly. It sets the minimum confirmation offset from all remote users in the **sr_queue_state** table.

sp_queue_set_progress procedure

Purpose

This procedure is used by the SQL Remote Message Agent, and should not be called directly.

Syntax

sp_queue_set_progress *page_id*, *row_id*, *commit_offset*, *backup_offset*, *marker*

Description

This procedure is used by the SQL Remote Message Agent, and should not be called directly. It sets the transaction log scanning progress value in the **sr_queue_state** table.

sp_queue_transaction procedure

Purpose

This procedure is used by the SQL Remote Message Agent, and should not be called directly.

Syntax

sp_queue_transaction *offset*, *user_id*

Description

This procedure is used by the SQL Remote Message Agent, and should not be called directly. It adds a new transaction to the stable queue.

sp_remote procedure

Purpose

This procedure is used by the SQL Remote Message Agent, and should not be called directly, with a single exception described below. It manages rows in the **sr_remoteuser** table.

Syntax

sp_remote *operation*, *user_name* [, *offset*] [, *confirm*]

Argument	Description
operation	The name of an action. The only value that should be used by a user is **reset**; all others are for use by the Message Agent.
user_name	The name of the remote user being reset
offset	Not used
confirm	Not used

Description

This procedure is used by the SQL Remote Message Agent, and should not be called directly with the single exception of the **reset** call. It maintains the message tracking information in the **sr_remoteuser** table.

The following special case can be used directly, when creating a custom database extraction process:

```
sp_remote reset, remote_user
```

where *remote_user* is the remote user name.

This command starts all subscriptions for a remote user in a single transaction. It sets the **log_sent** and **confirm_sent** values in **sr_remoteuser** table to the current position in the transaction log. It also sets the created and started values in **sr_subscription** to the current position in the transaction log for all subscriptions for this remote user. The procedure 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.

sp_remote_option procedure

Purpose

To set a SQL Remote option.

Syntax

sp_remote_option *option_name*, *option_value*

Argument	Description
option_name	The name of one of the SQL Remote options
option_value	The value to which the option is set.

See also

"SQL Remote options" on page 323.

Description

The SQL Remote options provide control over replication behavior. The following options are available in Adaptive Server Enterprise:

OPTION	VALUES	DEFAULT
Blob_threshold	**Integer, in K**	**256**
Compression	**-1 to 9**	**6**
Delete_old_logs	**ON**, **OFF**	**OFF**
Qualify_owners	**ON**, **OFF**	**OFF**
Quote_all_identifiers	**ON**, **OFF**	**OFF**
Replication_error	*procedure-name*	**NULL**
Subscribe_by_remote	**ON,OFF**	**ON**
Verify_threshold	*integer*	**256**
Verify_all_columns	**ON,OFF**	**OFF**

A complete description of these options is provided in "SQL Remote options" on page 323.

Example

- The following statement sets the Verify_all_columns option to OFF, so that old values of update statements applied by the Message Agent are not checked automatically for all columns.

```
sp_remote_option Verify_all_columns, OFF
go
```

sp_remote_type procedure

Purpose

To create or modify a SQL Remote message type.

Syntax

sp_remote_type *type_name publisher_address*

Argument	Description
type_name	The message type to create or alter. This must be one of the following: ♦ file ♦ ftp ♦ smtp ♦ mapi ♦ vim
publisher_address	The address of the publisher under the specified message type.

See also

"sp_drop_remote_type procedure" on page 393

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 registry entry, or of the current directory if that is not set.

For the FILE link, the procedure also causes the Message Agent to look for incoming messages in the address given for each message type.

Example

♦ The following example creates a FILE message type for a database, and gives the publisher's address as a subdirectory of the SQLREMOTE location named *publisher:*

```
sp_remote_type file, publisher
go
```

sp_remove_article procedure

Purpose

To remove an article from a publication

Syntax

sp_remove_article *publication_name, table_name*

Argument	Description
publication_name	The name of the publication from which the article is to be deleted.
table_name	The table containing the article.

See also

"sp_add_article procedure" on page 387

Description

Running **sp_add_article** removes an article from a publication. Any article including parts of the named table is removed from the publication.

Example

- The following statement removes any articles that use part of the **SalesRep** table from a publication named **SalesRepData**:

```
sp_remove_article SalesRepData, SalesRep
go
```

sp_remove_article_col procedure

Purpose

To remove a column from an article in a publication.

Syntax

sp_remove_article_col *publication_name, article_name, column_name*

Argument	Description
publication_name	The name of the publication to which the article belongs.
article_name	The article from which the column is to be removed.
column_name	The column to be removed from the article.

See also

"sp_add_article_col procedure" on page 388
"sp_remove_article procedure" on page 425

Description

You can remove a column from a publication using sp_remove_article_col.

To remove a column using sp_remove_article_col, the column must have been explicitly added to a publication using the "sp_add_article_col procedure" on page 388. Although the "sp_add_article procedure" on page 387, without use of sp_add_article_col, adds all the columns of a table to a publication, you cannot remove a single column from such a publication using sp_remove_article_col.

Example

- The following statement removes the column emp_lname of the employee table from a publication named Personnel:

```
sp_remove_article_col 'Personnel', 'employee',
'emp_lname'
go
```

sp_remove_remote_table procedure

Purpose

To mark a table as unavailable for SQL Remote replication.

Syntax

sp_remove_remote_table *table_name*

Argument	Description
table_name	The table to be marked as not available for SQL Remote replication.

See also

"sp_add_remote_table procedure" on page 389
"sp_modify_remote_table procedure" on page 401

Description

Marks a table as unavailable for replication. Once this procedure has been called, the data in the table cannot be shared with other databases using SQL Remote.

Example

- The following statement marks the **employee** table as unavailable for replication:

```
sp_remove_remote_table employee
go
```

sp_revoke_consolidate procedure

Purpose

To stop a user from being able to receive SQL Remote messages from this database.

Syntax

sp_revoke_consolidate *user_name*

Argument	Description
user_name	The user ID who will no longer be able to act as a consolidated database.

See also

"sp_grant_consolidate procedure" on page 395

Description

REMOTE permissions are required for a user ID to receive messages in a SQL Remote replication installation. The **sp_revoke_consolidate** procedure removes the consolidated database user ID from the list of users receiving messages from the current database.

Example

- The following statement revokes consolidated permissions from user **hq_user**:

```
sp_revoke_consolidate hq_user
go
```

sp_revoke_remote procedure

Purpose

To stop a user from being able to receive SQL Remote messages from this database.

Syntax

sp_revoke_remote *user_name*

Argument	Description
user_name	The user ID who will no longer be able to receive SQL Remote messages.

See also

"sp_grant_remote procedure" on page 397

Description

REMOTE permissions are required for a user ID to receive messages in a SQL Remote replication installation. The **sp_revoke_remote** procedure removes a user ID from the list of users receiving messages from the current database.

Example

- The following statement revokes remote permissions from user **Field User**:

```
sp_revoke_remote 'Field user'
go
```

sp_subscription procedure

Purpose

To manage subscriptions.

Syntax

sp_subscription *operation,*
publication_name,
user_name,
[*subscribe_by*]

Argument	Description
operation	The operation to be performed. This must be one of the following: ♦ **create** To create a subscription to a given publication for a user. ♦ **drop** To drop a subscription to a given publication for a user. ♦ **start** To start a subscription to the named publication. ♦ **stop** To stop a subscription to the named publication. ♦ **synchronize** To synchronize a subscription to the named publication.
publication_name	The name of the publication to which the subscription refers.
user_name	The user ID who is being subscribed to the publication.
subscribe_by	The subscription value.

See also

"Creating subscriptions" on page 198.

Description

The **sp_subscription** procedure is used to manage subscriptions. The first argument to the procedure (*operation*) specified whether the procedure is being created, dropped, started, stopped, or synchronized.

In general, starting and synchronizing subscriptions is done using the extraction utility. Sybase Central does not display subscriptions as started until the Message Agent first runs against the database.

Example

♦ The following statement creates a subscription for user **SalesRep1** to the **SalesRepData** publication, which has no subscription expression.

```
sp_subscription create,
   SalesRepData,
   SalesRep1
go
```