Enterprise SQL Server Manager™ Reference Manual

Enterprise SQL Server Manager Release 11.0

Document ID: 36704-01-1100-02

Last Revised: May 29, 1996

Principal authorship: Systems Management Products Group

Document ID: 36704-01-1100

This publication pertains to Enterprise SQL Server Manager Release 11.0 of the Sybase database management software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

Document Orders

Sybase Trademarks

Restricted Rights

Table of Contents

List of Tables

About This Book

Enterprise SQL Server Manager Reference Manual is a reference tool describing the basic syntax and usage information for every Enterprise SQL Server Manager™ command. It also explains the Enterprise SQL Server Manager command line interface and its conventions.

Audience

This manual is for administrators who want to manipulate Sybase® SQL Server™ database objects within the Tivoli Management Environment, either from the operating system command line or using scripts.

This manual is for experienced Enterprise SQL Server Manager administrators. It assumes you are familiar with SQL Server and database concepts and the procedures described in the *Tivoli Management Platform Guide* and the *Enterprise SQL Server Manager User's Guide*. You should read those books before using this manual.

How to Use This Book

Use this book as a reference tool for complete information on Enterprise SQL Server Manager command syntax and usage. This book consists of the following chapters:

- Chapter 1, "Introduction," gives an overview of the Enterprise SQL Server Manager command line interface and describes common command behavior and usage.
- Chapter 2, "Enterprise SQL Server Manager Commands," contains reference information on every Enterprise SQL Server Manager command. Commands appear in reference page format, listed alphabetically by name. There is also a summary of commands grouped by function at the beginning of the chapter.
- Appendix A, "Database Options," describes the SQL Server database options that Enterprise SQL Server Manager supports.
- The Glossary defines the special terms that appear in this book.

Related Documents

In addition to this document, the following manuals are part of the Enterprise SQL Server Manager documentation set:

- *Enterprise SQL Server Manager Release Bulletin* contains important release-specific information, including a list of known problems.
- *Enterprise SQL Server Manager Installation and Planning Guide* contains planning information and describes how to install Enterprise SQL Server Manager.
- *Enterprise SQL Server Manager User's Guide* is a guide to Enterprise SQL Server Manager and SQL Server systems management using Enterprise SQL Server Manager.
- *Tivoli Management Platform Release Notes* contain important release-specific information about the Tivoli Management Environment (TME) included with Enterprise SQL Server Manager.
- *Tivoli Management Platform User's Guide* is a comprehensive guide to the TME desktop.
- *Tivoli Management Platform Reference Manual* describes the TME command line interface.
- *Tivoli/Sentry User's Guide* describes the Tivoli / Sentry management application tool. Enterprise SQL Server Manager Event Monitoring (EMON) Services is based on the technology used by Tivoli / Sentry. The *Tivoli/Sentry User's Guide* provides information that will help you use EMON Services.

Other Documents

- Online help includes step-by-step instructions for every Enterprise SQL Server Manager procedure.
- *System Administration Guide* describes how to administer SQL Server and databases, independent of system administration tools such as Enterprise SQL Server Manager. It includes the Transact-SQL™ commands and system procedures used to perform the functions for which Enterprise SQL Server Manager provides its own interface.
- *SQL Server Reference Manual* provides basic syntax and usage information for every command, function, system procedure, and catalog stored procedure.

- *SQL Server Utility Programs* documents SQL Server utility programs, such as **bcp**, that you execute from the operating system command line.
- *Sybase Troubleshooting Guide* contains information that helps you prevent or quickly respond to trouble with SQL Server. It includes topics such as how to determine appropriate settings of configuration variables, how to manage transaction logs, how to develop good recovery procedures, and how to use disk mirroring effectively.

Other Sources of Information

Sybase offers a system and database administration class called "SQL Server Administration." For details, contact:

Mail:	Education Registrars Sybase Professional Services 77 South Bedford Street Burlington, MA 01803
Phone:	(800) 8-SYBASE or (617) 564-6970
Fax:	(800) 792-2733 or (617) 564-6960
E-mail:	registrars@sybase.com

Notational Conventions

The following sections describe the style and syntax conventions used in this book.

Style Conventions

Syntax statements (displaying the syntax and all options for a command) appear as follows:

sunmanageserver –name *server_name* [–*version*] [–help]

or for a command with too many options to fit on one line, as:

```
sgetroleaudit [–types role_names] [–onlynames]
   [–list] [–server server_name] [–version] [–help]
```

The multiline syntax format is for clarity of presentation in this manual. When you enter a command, do not press the Enter key (or Return key, depending on your keyboard) until you finish typing the entire command.

Examples showing the use of Enterprise SQL Server Manager commands appear as follows:

```
   sgetuser -database SMADA/pubs2
```

In this example, SMADA is the name of a SQL Server installation and *pubs2* is the name of a database in SQL Server. If an example is prefaced with 1., 2., and so on, this is the example number.

If a command example takes several lines, some of which must be ended with a control key, the control key is indicated as follows:

```
   scrtrule -name quiz_answer
     -database PARIS_DEV/master <Return>
   answer in ("A", "B", "C") <Return>
   <CTRL-D>
```

Examples of output from Enterprise SQL Server Manager commands appear as follows:

```
   *** Users In Database: SMADA/pubs2 ***
   User        Login       Group
   ----------  ----------  ----------
   cat         cata        public
   cootie      cwilliams   public
   guest       null        public
   hjones      hjones      public
   mercer      mercere     public
   example%
```

Syntax Conventions

In this book, all command syntax statements use the following notational conventions:

Table 1: Enterprise SQL Server Manager command syntax conventions

Example	Description
`scrtlogin`	Command keywords appear in lowercase.
name	Variable arguments (words that stand for values that you supply in the command) appear in italic.
`[-help]`	Brackets mean that including the enclosed items in the command is **optional**. Do not include the brackets in your command.
{*name* [-wildcard]}	Braces mean that you **must** include one or more of the enclosed items. Do not include the braces in your command.
`true | false`	A vertical bar means that you can select only one of the two items that appear. Do not include the vertical bar in your command.
`[, device size]...`	Ellipses (...) indicate that you can repeat the preceding item as many times as you like in the command. Do not include the ellipsis in your command.

Include all other characters (commas, parentheses, and others) just as they appear in the command syntax statement.

If You Need Help

Help is available for the Enterprise SQL Server Manager user in the form of documentation, online help, online manual pages, and the Sybase Technical Support Center.

Technical Support

Each Sybase installation site has one person (or more) designated to contact the Technical Support Center. If you cannot solve a problem using the manuals or online help, ask the designated person at your site to contact the Technical Support Center for help.

1 Introduction

In addition to its graphical user interface (GUI), Enterprise SQL Server Manager includes a rich set of commands that let you manipulate Enterprise SQL Server Manager resources from the operating system command line or from within scripts.

Scripts that combine Enterprise SQL Server Manager commands, Tivoli Management Environment (TME) commands, and virtually any scripting language (Bourne shell, C shell, Korn shell, Perl, and others) can be powerful tools for batch processing repetitive or large-scale SQL Server management chores. Furthermore, you can define these scripts as tasks in the Tivoli Management Platform (TMP) task library for use by other administrators, and schedule them as jobs in the TMP scheduler. See *Tivoli Management Platform User's Guide* for more on the TMP task library and scheduler.

The Enterprise SQL Server Manager Command Line Interface

The Enterprise SQL Server Manager command line interface (CLI) follows a consistent and logical syntax that makes it easy to remember and use Enterprise SQL Server Manager commands. This section outlines the syntax conventions common to all Enterprise SQL Server Manager commands. For a description of the notational conventions that this document uses, see "Notational Conventions" on page xviii.

Command Names

All Enterprise SQL Server Manager command names begin with the prefix "s" to insure uniqueness and to identify them as Sybase commands. A verb usually follows the "s" prefix. The most common verbs are:

Table 1-1: Command verbs

Verb	Action
comp	compare
copy	copy
crt	create
del	delete
get	get
set	set

An object type follows the verb and indicates the kind of object the command manipulates or accesses. Object types include **login**, **server**, **db** (for "database"), **dev** (for "device"), and others.

Understanding this convention, you could guess that a command named:

```
scrtlogin
```

creates a SQL Server login.

Command Options

Command options modify the behavior of a command and specify the values you want a command to use.

- All command options begin with a dash (–).
- The CLI uses multiple-character option names (instead of single-character, UNIX-like mnemonic "switches") to make command syntax more descriptive and intuitive. For example, **–name**, **–server**, and **–database**.
- Command options can occur in any order on the command line.
- You can enter any unambiguous substring of an option name. For example, you can shorten **–name** to **–na** or **–n** if no other options in the command begin with the letters "na" or "n."
- Command options are not case-sensitive (**–name**, **–Name**, and **–NAME** represent the same option).

Option Values

Many command options also require values. These values can be predefined keywords like **TRUE** or **ENABLE**, numbers, or names that identify servers, databases, server logins, and other objects. Also, some command options accept lists of values. The following sections describe the different kinds of option values the CLI accepts and the rules for using them.

Keywords

Like command options, most keyword values (such as **primary**, **secondary**, **true**, and **disable**) are not case-sensitive, and unambiguous substrings for keywords are acceptable (for example, **tr** for **true**).

SQL Server Keywords and Object Names

SQL Server keywords (such as **sa_role**, **statistics io**, and so forth) are case-sensitive, and you must specify them exactly as they are known to SQL Server. Also, you must enter most SQL Server keywords in their entirety—substrings are illegal.

SQL Server object names, such as login and server names, are also case-sensitive. Here, too, you must enter complete object names—substrings are illegal. The names must comply with the SQL Server rules for identifiers (see *SQL Server Reference Manual*). Most commands use **–name** and **–names** options to specify object names.

Boolean Keywords

Boolean option values produce a value of 0 or 1. You can use any of the Boolean keywords listed in Table 1-2 with most command options that accept a Boolean value:

Table 1-2: Boolean keywords

1	0
true	false
yes	no
on	off
enable	disable
1	0

The exception to this is when a Boolean option value is serving as a SQL Server keyword (for instance, in enabling or disabling a SQL Server database option). In these cases, you must supply the Boolean keyword exactly as specified in the Enterprise SQL Server Manager command syntax.

Numbers

Numeric values can include the digits 0 through 9. Thus, numeric option values must always be positive integers.

Option Lists

Some command options accept only a single value; others accept a list of items. For example, most **sget**, **scopy**, and **scomp** commands have a **–names** option that accepts a list of one or more object names. When a command option accepts a list of items, the following rules apply:

- You must separate the items in the list with commas, or with one or more spaces, or both. For example:

  ```
  -roles sa_role,oper_role
  ```

  specifies an option value list with two items: **sa_role** and **oper_role**.

- If a list contains one or more spaces, you must enclose the list in double quotation marks. For example:

  ```
  -roles "sa_role oper_role"
  ```

  specifies a list with two items separated by a space: **sa_role** and **oper_role**. Because the list contains a space, the list is enclosed in double quotation marks.

- When an item within a list contains multiple elements (for example, a list of devices and their corresponding sizes), you must separate each multi-element item with a comma, and you must separate each element within an item with a space. For example:

  ```
  -devices "default 4, db_dev 2"
  ```

  specifies a list with two items (*default 4* and *db_dev 2*), where each item contains two elements (a device name and a size).

- When an element within a list item contains a space, the element must be enclosed in single quotation marks. For example:

  ```
  -config "'recovery interval' 3"
  ```

  specifies a list item with two elements: the configuration variable **recovery interval** and the value 3. Because **recovery interval** is a list element that contains a space, it is enclosed in single quotation marks.

Wildcards

You can often use wildcards in option value lists that specify SQL Server object names. **Wildcards** are special characters that represent one or more characters. Any character or set of characters can replace a wildcard character. Wildcards combined with other characters form a pattern-matching string to save time when referring to a large group of names. To enable wildcard matching for names listed in a command, use the **–wildcard** option. For example, suppose you want to specify all database user names beginning with "w." Instead of typing each name in the list, one by one, as in:

```
-names "woodyard, woode, williams"
```

you can enter a pattern-matching string that contains the percent sign (%) wildcard character to represent zero or more characters:

```
-names w% –wildcard
```

The pattern “w%” specifies any names that begin with “w” followed by zero or more characters. Table 1-3 lists the wildcards that Enterprise SQL Server Manager accepts:

Table 1-3: SQL Server wildcards

Wildcard	Meaning
`%`	Any string of zero or more characters.
_ (underscore)	Any single character.
[*chars*]	Any single character in the list or range of characters *chars*. A list is any string of characters. For example, [a2br] matches the single characters “a”, “2”, “b”, or “r”. Note that the lists [abcdef] and [fcbdae] match the same set of characters. You can use a dash (*char1–char2*) to specify a range of characters. All characters in the sort order between *char1* and *char2* (inclusive) match the wildcard. For example, [A-Za-z] matches any alphabetic character in the 7-bit ASCII sort order.
[^*chars*]	Matches any single character **not** in the specified list or range. For example, [^a-c] matches any single character that is not “a”, “b”, or “c” (in the 7-bit ASCII sort order).

To use a wildcard character literally, enclose it in square brackets ([ ]). For example:

```
–names "user[_][a–c]" -wildcard
```

matches “user_a” and “user_b” but not “user_a.” The following are some more examples of using wildcards in option lists:

1. **`sgetlogin -names j% -wildcard`**

   Gets information on all logins beginning with the letter “j.”

2. **`sgetlogin -names s_ -wildcard`**

   Gets information on all two-character logins beginning with “s.”

3. **`sgetlogin -names "guest [AB]%" -wildcard`**

   Gets information on the login “guest” and all login names that begin with “A” or “B.”

4. **`sgetlogin -names "guest %[0-9]" -wildcard`**

   Gets information on the login “guest” and all login names ending in a digit.

Optional Values and Defaults

Most commands accept one or more options. Some options are mandatory; you must include them for the command to work successfully. In many cases, if you do not include an option, the command uses a default value. For example, many commands accept a **–server** option that specifies the managed server in which you want the command to work. If you omit this option, the command uses the default value (in this case, the current managed server). Surrounding brackets (“[“ and “]”) in the command syntax identify optional command elements.

Common Command Behavior

All Enterprise SQL Server Manager commands share the following behavior:

- The CLI reads input from standard input (*stdin*, file descriptor 0), sends normal command output to standard output (*stdout*, file descriptor 1), and sends error messages to standard error (*stderr*, file descriptor 2).
- All commands return an exit status: zero indicates success, non-zero indicates an error occurred.
- All commands display a usage message in response to syntax errors, missing command options, and the **–help** option.

scrt (Create) Commands

The **scrt** commands make new objects, such as SQL Server logins, databases, or schema objects. The **scrt** commands usually create one object at a time.

scomp (Compare) Commands

The **scomp** commands compare resources and objects between servers or databases and display any differences. To differentiate between the servers or databases you are comparing, one is called the “source” and the other is the “target.”

When identifying a server or database for comparison, most **scomp** commands accept standard **–source** *collection_name* and **–target** *collection_name* options to specify source and target servers or databases. Because both options default to the current server or database collection, you must supply at least one of the two names, and that name must be different from the current server or database collection. If you omit both options, the **scomp** command fails and displays an error message.

In the output of **scomp** commands, lines beginning with < refer to the source and lines beginning with > refer to the target. For example:

```
scomplogin –source AGRA –target DEPK
```

```
<<<source: AGRA
>>>target: DEPK
----------bmiley----------
Lock Status:
< ON
> OFF
----------cwilliams----------
Default Database:
< master
> pubs2
```

This output shows that the login *bmiley* is locked in the source server AGRA, but the same login is not locked in the target server DEPK. Likewise, the default database for the login *cwilliams* is *master* in the server AGRA, but *pubs2* in the server DEPK.

If an attribute value exists in the source or target and not in its counterpart, the **scomp** commands display only the attribute name and value for the existing attribute. There are no lines beginning with < or > for the nonexistent attribute. This behavior is similar to the UNIX **diff** command. For example:

```
scomplogin –name jhodges –source AGRA –target DEPK
```

```
<<<source: AGRA
>>>target: DEPK
---------------------------------------------------
---------- jhodges ----------
Login Full Name:
< Johnny Hodges
Login Default Database:
< master
Default Language:
< us_english
Login Lock:
< 0
Login Audit Flags:
< 0
Login Password Date:
< June 29, 1994 7:03:26 pm
Login Expiration Date:
< null
Connected:
< null
```

This output shows that the login *jhodges* exists only on the source server AGRA.

Similarly, if the object does not exist in either source or target, there are no lines beginning with < or > in the output for that object. For example:

```
scomplogin –name rnance –source AGRA –target DEPK

<<<source: AGRA
>>>target: DEPK
----------------------------------------------------
No differences
```

This output shows that the login *rnance* is the same on each server or that it does not exist on either managed server.

scopy (Copy) Commands

The **scopy** commands copy resources and objects between servers or databases. Typically, you cannot customize copy operations—you cannot change object attributes during a copy operation (the **scopylogin** and **scopydb** commands are exceptions—**scopylogin** lets you specify a password to change and **scopydb** lets you override data and log device allocations). You can use the **sset** commands for this purpose.

In a copy operation, the server or database from which you are copying is called the “source,” and the server or database to which you are copying is called the “target.” All copy operations create a new object in the target.

Like the **scomp** commands, you use standard **–source** *collection_name* and **–target** *collection_name* options to specify source and target servers or databases. Because both options default to the current server or database collection, you must supply at least one of the two collection names, and that name must be different from the current server or database collection.

sdel (Delete) Commands

The **sdel** commands permanently remove an object—usually one object at a time. This operation deletes the object from SQL Server and from the TME policy region in which it is a managed resource.

When you delete an object, Enterprise SQL Server Manager checks whether that object is in a profile. If it is, the object is marked as deleted and the next time the profile is distributed, the object is deleted from the subscribing SQL Server.

sget (Get) Commands

The **sget** commands retrieve and display information about object attributes. Examples of object attributes are SQL Server configuration variables, names of related objects, and their values.

Tabular and List Output Format Options

By default, **sget** commands generate summary reports that list key object attributes. You can generate summary reports in either a tabular format (the default) or list format. For example, the following command generates a summary report in tabular format:

```
sgetuser -database KOKO/pubs2
```

```
*** Users In Database: KOKO/pubs2 ***
User        Login       Group
----------  ----------  ----------
cat         cata        public
cootie      cwilliams   public
guest       null        public
hjones      hjones      public
mercer      mercere     public
```

You can use the **–list** option to display summary information in list format instead of table format. The following command generates the same summary report in list format:

```
sgetuser –database KOKO/pubs2 –list

*** Users In Database: KOKO/pubs2 ***
User Name in Database: cat
Login Name: cata
Group Name: public

User Name in Database: cootie
Login Name: cwilliams
Group Name: public

User Name in Database: guest
Login Name: null
Group Name: public

User Name in Database: hjones
Login Name: hjones
Group Name: public

User Name in Database: mercer
Login Name: mercere
Group Name: public
example%
```

Only summary reports can display output in tabular format. When you specify detail output options in an **sget** command, the output appears in list format. The next section describes detail output options.

Attribute Summary and Detail Output Options

Because some objects have many attributes, summary reports simplify the output by automatically listing the key attributes of an object. Additionally, many **sget** commands accept options to display specific information in the output. In most cases, these detail options restrict the output to the attributes specified in the command. For example, the following command uses the **–aliases** option to get only user aliases:

```
sgetuser -data KOKO/pubs2 -aliases

*** Users In Database: KOKO/pubs2 ***
User Name in Database: cat
Aliases:
    Login Name: testuser10
    Login Name: testuser11

User Name in Database: cootie
Aliases:

User Name in Database: guest
Aliases:

User Name in Database: hjones
Aliases:

User Name in Database: mercer
Aliases:
```

Many **sget** commands include a **–summary** option so you can retrieve the standard summary information with the detail information:

```
sgetuser –database KOKO/pubs2 –summary –aliases

*** Users In Database: KOKO/pubs2 ***
User Name in Database: cat
Login Name: cata
Group Name: public
Aliases:
    Login Name: testuser10
    Login Name: testuser11

User Name in Database: cootie
Login Name: cwilliams
Group Name: public
Alias Table:

User Name in Database: guest
Login Name: null
Group Name: public
Alias Table:
```

```
User Name in Database: hjones
Login Name: hjones
Group Name: public
Alias Table:

User Name in Database: mercer
Login Name: mercere
Group Name: public
Alias Table:
```

Capturing and Viewing Report Information

Because the CLI sends informational reports to standard output (*stdout*), you can use standard UNIX input/output redirection operators (<, >, >>, and |) to capture output in files or to send the output to other commands.

➤ ***Note***

scheckindex and **schecktable** send output to *stderr*, not *stdout*.

sset (Set) Commands

The **sset** commands modify the current attributes of an object—usually one object at a time. When you change the attributes of an object, Enterprise SQL Server Manager checks whether the object is part of a profile and whether it is subject to validation policy. If it is, Enterprise SQL Server Manager implements the policy. See *Enterprise SQL Server Manager User's Guide* and *Tivoli Management Platform User's Guide* for information about validation policy.

Using Enterprise SQL Server Manager Commands

Enterprise SQL Server Manager commands are fully integrated with Tivoli Management Environment (TME) commands and Tivoli Management Platform (TMP) services. For example, rather than executing Enterprise SQL Server Manager commands exclusively, it is more often the case that you use Enterprise SQL Server Manager commands with TME commands to navigate between collections and manipulate resources. The TME commands that navigate between collections (**wcd**, **wls**, **wpwd**, and others) provide a context in which Enterprise SQL Server Manager commands manipulate SQL Server and database collection objects.

In creating and modifying a resource, those operations are subject to any default and validation policy for that resource. Be aware of how default and validation policy impact the Enterprise SQL Server Manager commands you execute.

Also, Enterprise SQL Server Manager commands use TMP transaction services to ensure data consistency. It is important to understand how Enterprise SQL Server Manager commands execute as transactions.

The following sections describe how Enterprise SQL Server Manager commands work with TME commands and TMP services in the Sybase Enterprise Management Architecture. See *Tivoli Management Platform User's Guide* for specific information on TME commands and TMP services.

Command Context

When you are using the graphical user interface (GUI), Enterprise SQL Server Manager determines the context (the server or database collection object to use) for an operation from the object and collection currently selected. When using Enterprise SQL Server Manager commands, you can set the context in one of two ways:

- Explicitly naming server and database objects in Enterprise SQL Server Manager commands, or
- Setting the current working collection in the Tivoli Management Environment (TME) to the desired collection object and using that as the default in Enterprise SQL Server Manager commands.

Thus, the following two scenarios are equivalent:

1. **`scrtlogin -name cwilliams -password MadnessInGreat1s -server DEPK`**

   Creates a SQL Server login "cwilliams" in the managed server DEPK by identifying the server explicitly in the **scrtlogin** command.

2. **`wcd /Library/PolicyRegion/`*`policy_region_name`*`/DEPK`**
   `scrtlogin -name cwilliams -password MadnessInGreat1s`

   Uses the TME **wcd** command to change the current working collection to the DEPK server collection object, and then uses the Enterprise SQL Server Manager **scrtlogin** command to create a login in the current collection (DEPK). Use of a full path name is recommended.

Because a managed SQL Server or database resource is a collection object, you can refer to it in TME commands using standard collection path name conventions (see *Tivoli Management Platform User's Guide* for TME collection navigation commands and path naming conventions). When naming SQL Server or database collection objects explicitly in Enterprise SQL Server Manager commands, the following conventions apply.

SQL Server Names

Many commands accept a **–server** option that specifies the server collection in which you want the command to work. In the Enterprise SQL Server Manager object hierarchy, all server names are unique, and there is no need to supply a collection path using a slash (/) in the option value. In fact, doing so usually produces an error.

Database Names

The **–database** options specify database collection objects, which exist in server collections. Database names need not be unique from server to server (the same database names may occur in several server collections). However, database names must be unique within the same server. Therefore, in supplying **–database** option values, you can specify either a database name alone or a database name preceded by a server collection name and a slash (/), as in the following example:

```
AGRA/pubs2
```

This example identifies the *pubs2* database in the server collection named AGRA. If you do not specify a server collection, the command assumes the database exists in the current collection. If the current working collection is not a server collection (or a user-defined collection within a server collection) and you specify a database name alone, the command displays an error message.

Profile Management

To distribute profiles from the command line, use the Tivoli command **wdistrib**. If you use **wdistrib** with Enterprise SQL Server Manager commands, you must use the **–m** argument. You can also use the other Tivoli commands for profile management.

Default and Validation Policy

To implement policy from the command line, use the Tivoli commands for enabling policy. Default policies are not applied to objects created with Enterprise SQL Server Manager commands. Validation policies are applied to changes made using an **sset** command if the object is part of a profile and is subject to validation policy. If it is, Enterprise SQL Server Manager implements the policy.

For more information on default and validation policy:

- See *Tivoli Management Platform User's Guide* for more information about defining and assigning default and validation policies.
- See *Enterprise SQL Server Manager User's Guide* for more information about Enterprise SQL Server Manager default and validation policy.

Enterprise SQL Server Manager Commands and Transactions

When you execute an Enterprise SQL Server Manager command, it runs as a transaction. When a command runs as a transaction, operations on all objects must complete successfully for the command as a whole to complete successfully. If an operation fails on a particular object, the command rolls back all changed objects to their original states.

For example, if you execute a command such as:

```
scopylogin -names “bwebster barneyb harryc” -target CONCORD
```

and the copy operation is successful for “bwebster” and “barneyb”, but fails for “harryc”, the successful changes are rolled back, as well as any changes made in the effort to copy “harryc”. The result is that no changes are made to CONCORD as a result of this command.

By contrast, if you execute three separate commands (not in a script), such as:

```
scopylogin -names bwebster -target CONCORD
scopylogin -names barneyb -target CONCORD
scopylogin -names harryc -target CONCORD
```

and the copy operation for “harryc” fails, only changes made trying to copy “harryc” are rolled back. The copy operations for “bwebster” and “barneyb” are in separate commands, and, if successful, they are committed. For information about how Enterprise SQL Server Manager commands work in scripts, read “Using Enterprise SQL Server Manager Commands in Scripts”.

Error Handling

The CLI applies the following error handling for all Enterprise SQL Server Manager commands:

- When syntax errors occur, the CLI displays a usage message for the command.
- When errors originate from SQL Server, the CLI reports the SQL Server-generated messages in their original form.

Role Requirements

Each Enterprise SQL Server Manager command requires that an administrator have the correct combination of authorization roles to successfully use the command. There are three kinds of authorization roles:

- **TME roles** are the basic Tivoli administrator authorization roles included with TME.
- **ESSM roles** are the application-specific administrator authorization roles that Enterprise SQL Server Manager adds to TME.
- **SQL Server roles** are the administrative roles available in SQL Server for SQL Server logins.

You assign TME roles and Enterprise SQL Server Manager roles to TME administrators, and you assign SQL Server roles to the SQL Server logins for TME administrators. The following table lists the roles in each category:

Table 1-4: Three kinds of administrator roles

TME roles	ESSM roles	SQL Server roles
super	server	sa_role
senior	security	sso_role
admin	space	oper_role
user	dump	
restore	load	
backup	schema	
install_product	monitor	
install_client	cache	

Notes

- To manipulate objects (create, delete, modify, and so on) that are not owned by "dbo", you must either be the Database Owner or have "sa" role.
- To manipulate objects in a profile manager, you need the role documented for the command, plus **server** role.
- There are no commands for working with named caches. To work with named caches, use the graphic user interface.

Manual Page Conventions

The manual page for each Enterprise SQL Server Manager command contains a "Permissions" section that lists the required authorization roles from each category. In addition to listing specific roles, two other keywords may appear in the role requirement listings:

- When the word "**none**" appears, you do not need any of the roles from the category to run the command.
- When the word "**any**" appears, it means that an administrator must have at least one of the roles from the category, but it does not matter which role. You need at least the Tivoli **user** role to run most commands.

For More Details

The following table lists other sources of information about authorization roles:

Table 1-5: For more information on authorization roles

For more on...	See...
TME roles	Chapter 3, "Tivoli Administrators," in *Tivoli Management Platform User's Guide*
ESSM roles	Chapter 7, "Getting Started," in *Enterprise SQL Server Manager Installation and Planning Guide.*
Assigning TME and ESSM roles to administrators	Chapter 3, "Tivoli Administrators," in *Tivoli Management Platform User's Guide*
SQL Server roles	*System Administration Guide*

Using Enterprise SQL Server Manager Commands in Scripts

You can use Enterprise SQL Server Manager commands in scripts and run them as Tivoli tasks. To run tasks, a Tivoli administrator must have both a user and group specified. You specify user and group when you create an administrator. You can change the group specification on the Administrator Properties dialog box. For more information about creating administrators and for information about how to create and run a task, see *Tivoli Management Platform User's Guide*.

Identifying Endpoints in Scripts

If you want to run a task on multiple managed resources and the task needs to know the name of the resource on which it is running, you can use the $ENDPOINT variable in the task script. Enterprise SQL Server Manager endpoints use the following syntax:

- SQL Server - *server_name* (ManagedSQLServer)
- Database - *server_name db_name* (ManagedSQLDatabase)

As an example, suppose you want to back up all the databases that subscribe to a profile manager. You could write an individual script for each database in the following format and specify each database and SQL Server in each script.

```
#!/bin/sh
sdumpdb -name db_name -server server_name << EOF
-stripedevice tapedump1 -backupserver SYB_BACKUP
-dumpvolume pvolume
EOF
```

Or, you can write one script using *$ENDPOINT*, where *$ENDPOINT* is the object on which the task is executing.

```
#!/bin/sh
set `echo $ENDPOINT`
SERVER_NAME=$1
DB_NAME=$2
OBJ_TYPE=$3
# Check to see if this is a database endpoint
if [ "$OBJ_TYPE" != "ManagedSQLDatabase" ] ; then
```

```
  echo "The object to which you are trying to
distribute is not a database."
  exit 1
fi
sdumpdb -name $DB_NAME -server $SERVER_NAME << EOF
-stripedevice tapedump1 -backupserver SYB_BACKUP
-dumpvolume pvolume
EOF
```

For each Execution Target (endpoint) selected on the Execute Task dialog box, the script identifies the name of the object and uses it as input to the **sdumpdb** command.

You can execute a task against the following managed resources:

- Managed SQL Servers
- Databases that are subscribers to and sources for Database Profile Managers

If you want to execute tasks on a database that is not a subscriber to or a source for a Database Profile Manager, you can register the database as a managed resource with the **wregister** command. For the name parameter, use the syntax “*server_name database_name*”. For example:

```
wregister -r ManagedSQLServer “ROME pubs2”
```

For information about the **wregister** command, see *Tivoli Management Platform Reference Manual*.

Script Strategies

If you define a script as a task in the TMP Task Library, the behavior of the task depends on how you write the script. You can write a script to behave in the following ways:

- Execute all commands in the script and track status. If any command in the script fails, the script as a whole fails and all changes to affected SQL Server installations roll back.
- Execute all commands in the script. If any command in the script fails, no changes are made as a result of that command, but the script continues and changes made by successful commands are maintained. This type of script is called “best effort”.

➤ ***Note***

In the context of the previous discussion of Enterprise SQL Server Manager commands and transactions, in the first type of script the entire script is a transaction. In the second type, the script is not a transaction. However, the individual commands in the script are treated as transactions, as described above.

The following examples illustrate the types of scripts described in this section:

Example 1: Complete the script, but roll back all changes if any command fails

```
#!/bin/sh
# Initialize a variable for tracking status.
STATUS=OK
# Create logins on district SQL Servers
scrtlogin -name sales001 -password init_pass -server DISTRICT1
if [ "$?" != "0" ] ; then
  STATUS=ERROR
  echo "Error occurred adding login to SQL Server DISTRICT1" >&2
fi
scrtlogin -name sales001 -password init_pass -server DISTRICT2
if [ "$?" != "0" ] ; then
  STATUS=ERROR
  echo "Error occurred adding login to SQL Server DISTRICT2" >&2
fi
# Check to see if any failures occurred in script and exit
accordingly.
if [ "$STATUS" != "OK" ] ; then
    wtaskabort
fi
```

➤ ***Note***

wtaskabort is a Tivoli command. It is not documented in the *Tivoli Management Platform Reference Manual*. This command rolls back all changes made in the transaction.

Example 2: Complete the script, allow successful changes to stand

This script provides no feedback if a command fails, but you could add language similar to the **if** statement in Example 1 to send a message if a command fails.

```
#!/bin/sh
# Create logins on district SQL Servers
scrtlogin -name sales001 -password init_pass -server DISTRICT1
scrtlogin -name sales001 -password init_pass -server DISTRICT2
scrtlogin -name sales001 -password init_pass -server DISTRICT3
scrtlogin -name sales001 -password init_pass -server DISTRICT4
exit 0
```

Creating Nested Transactions in Scripts

If you run a script as a Tivoli task, you can use the Tivoli command **wruntask** to create nested transactions. The **wruntask** argument **–T** allows you to specify the transaction type of a task. Used in combination with the scripting strategies discussed in preceding sections, you can create a main task that uses **wruntask** to call a series of subtasks and precisely control the rollback behavior. See *Tivoli Management Platform Reference Manual* for information about **wruntask**.

Executing Tivoli Tasks with Enterprise SQL Server Manager

You can execute a Tivoli task the following ways:

- From the command line
- From the Tivoli Execute Task dialog box
- Using drag and drop.

For information about running a task from the command line or using the Execute Task dialog box, see *Tivoli Management Platform User’s Guide*.

You can execute a task by dragging its icon from a Task Library onto the icon for a Managed SQL Server, a SQL Server Profile Manager, or a Database Profile Manager. If you execute a task using drag-and-drop on a Managed SQL Server, it executes on the management host.

If you execute a task using drag-and-drop on a SQL Server Profile Manager or Database Profile Manager, it executes on the management hosts for all subscribers to the profile manager.

If you execute a task using the Execute Task dialog box, you can specify databases as endpoints. If you use drag-and-drop, you cannot choose databases to be endpoints, because they do not have icons in a policy region.

2 Enterprise SQL Server Manager Commands

This chapter describes each ESSM command. The commands are in alphabetical order. The following tables summarize the commands by function.

Server Management

Table 2-1: Server management commands

Command	Synopsis	Page
smanageserver	Registers a SQL Server as a managed resource in the TMR.	2-303
sunmanageserver	Removes the registration of a SQL Server as a managed resource in the TMR.	2-378
scompserver	Compares the configuration variables between managed SQL Servers and displays the differences.	2-68
sgetserver	Gets information about a managed SQL Server.	2-271
ssetserver	Changes the configuration for a managed SQL Server.	2-353
sstartserver	Starts a managed SQL Server.	2-370
sstopserver	Shuts down a managed SQL Server.	2-372
sgetprocess	Gets information about processes for a managed SQL Server.	2-255
skillprocess	Kills a process in a managed SQL Server.	2-298

Security Management

Enterprise SQL Server Manager includes commands for three areas of security management:

- Authentication—creating and verifying the identity of a login, database user, or remote SQL Server
- Authorization—managing group and user access to SQL Server objects and commands
- Auditing—recording security-related SQL Server and database activity in a traceable audit trail

Authentication

Enterprise SQL Server Manager has authentication management commands for:

- SQL Server login management
- Database user management
- Remote SQL Server management

SQL Server Login Management

Table 2-2: Login management commands

Command	Synopsis	Page
scrtlogin	Creates a SQL Server login in a managed SQL Server.	2-144
scomplogin	Compares logins between managed SQL Servers and displays the differences.	2-50
scopylogin	Copies logins from one managed SQL Server to another.	2-98
sdellogin	Deletes a login from a managed SQL Server.	2-190
sgetlogin	Gets information about logins in a managed SQL Server.	2-239
ssetlogin	Sets the configuration of a login in a managed SQL Server.	2-332
ssetsybaselogin	Sets SQL Server logins and passwords for an administrator.	2-358
sgetsybaselogin	Provides information about SQL Server logins and passwords for an administrator.	2-282
sdelsybaselogin	Deletes SQL Server logins and passwords for an administrator.	2-202
scrtsybaselogin	Creates SQL Server logins and passwords for an administrator.	2-164

Database User Management

Table 2-3: Database user management commands

Command	Synopsis	Page
scrtuser	Creates a user in a database so that the associated login has access to it.	2-173
scompuser	Compares users between databases and displays the differences.	2-74
scopyuser	Copies users from one database to another database.	2-118
sdeluser	Deletes a user from a database so that the associated login no longer has access to the database.	2-208
sgetuser	Gets information about users that have access to a specific database.	2-292
ssetuser	Sets a database user's attributes.	2-368

Remote Server Management

Table 2-4: Remote server management commands

Command	Synopsis	Page
scrtrmtserver	Creates a remote SQL Server entry in the *sysservers* table in a local managed SQL Server so that the remote SQL Server can start remote procedure calls (RPCs) on the local SQL Server.	2-157
scomprmtserver	Compares remote SQL Server attributes on one managed SQL Server with remote SQL Server attributes on another managed SQL Server. Optionally compares remote logins.	2-58
scopyrmtserver	Copies remote SQL Server attributes from one managed SQL Server to another. You can also use **scopyrmtserver** to copy remote logins.	2-105
sdelrmtserver	Deletes a remote SQL Server from the remote access configuration of a local managed SQL Server.	2-197
sgetrmtserver	Gets information about the remote SQL Server configuration for a specified local managed SQL Server.	2-259
ssetrmtserver	Sets the remote SQL Server configuration for a specified local SQL Server and remote SQL Server pair.	2-345
scrtrmtlogin	Creates remote logins in a remote server that are mapped to logins in a local managed SQL Server.	2-154
sdelrmtlogin	Deletes remote logins from a remote SQL Server.	2-195
ssetrmtlogin	Sets remote login configuration options.	2-343

Authorization

Table 2-5: Authorization management commands

Command	Synopsis	Page
scrtgroup	Creates a group in a specified database.	2-140
scompgroup	Compares groups between databases and displays the differences.	2-45
scopygroup	Copies groups from one database to another database.	2-94
sdelgroup	Deletes a group from a database.	2-186
sgetgroup	Gets information about groups in a database.	2-235
ssetgroup	Sets a group's user membership.	2-327
sgetobjperm	Gets information about current object permissions defined in a database.	2-245
ssetobjperm	Sets the permissions defined on a database object for users, groups, or roles.	2-338
sgetcmdperm	Gets information about current command permissions defined in a database.	2-214
ssetcmdperm	Sets the permissions defined on a command in a database for users, groups, or roles.	2-308

Auditing

Table 2-6: Auditing management commands

Command	Synopsis	Page
sinstallaudit	Installs the audit system on a managed SQL Server.	2-296
sgetloginaudit	Gets information about the auditing configuration of logins in a specified SQL Server.	2-243
ssetloginaudit	Sets auditing of one or more login accounts in a managed SQL Server.	2-335
sgetserveraudit	Gets information about the auditing configuration of server events in a managed SQL Server.	2-280
ssetserveraudit	Sets system-wide auditing and server event auditing in a managed SQL Server.	2-355
sgetroleaudit	Gets information about audited roles in a managed SQL Server.	2-262
ssetroleaudit	Sets auditing privileged command use for roles in a managed SQL Server.	2-347
sgetdbaudit	Gets information about the command auditing configuration of databases in a managed SQL Server.	2-224
ssetdbaudit	Sets command auditing of one or more databases in a managed SQL Server.	2-320
sgetprocaudit	Gets information about the configuration of execution auditing for stored procedures and triggers in a database.	2-253
ssetprocaudit	Sets auditing of stored procedure and trigger usage in a database.	2-340
sgettableaudit	Gets information about the configuration of table and view auditing for a particular database.	2-287
ssettableaudit	Sets auditing of accesses to tables and views in a database.	2-363
scrtauditrec	Adds a record (comment) to a SQL Server audit database.	2-123

Space Management

ESSM includes commands for three areas of space management:

- Database management
- Space allocation management
- Profile management

Database Management

Table 2-7: Database management commands

Command	Synopsis	Page
scheckdb	Invokes **dbcc** to check the consistency of a database.	2-19
scrtdb	Creates a new database in a managed SQL Server and registers the database as a managed resource in the Sybase Enterprise Management Architecture.	2-129
scompdb	Compares configuration options between databases and displays the differences. Also compares summary, space, and device information.	2-33
scopydb	Copies one database to another database, on the same or a different managed SQL Server.	2-83
sdeldb	Deletes a database from a specified managed SQL Server. Deletes all objects in the database from the SQL Server and frees the storage space allocated for it.	2-179
sgetdb	Gets information about databases in a managed SQL Server.	2-218
ssetdb	Sets the configuration of a database in a SQL Server, including its ownership, size, and configuration options.	2-316
scheckdb	Checks the logical and physical consistency of a database and displays the results.	2-19
sdumpdb	Backs up a database or transaction log.	2-211
sloaddb	Restores a database or transaction log.	2-300
schangevol	Notifies the Backup Server that the operator performed the requested volume handling during a dump or load.	2-17

Space Allocation Management

Table 2-8: Database device management commands

Command	Synopsis	Page
scrtdev	Creates a new database device in a managed SQL Server.	2-135
scompdev	Compares database devices between managed SQL Servers and displays the differences.	2-38
scopydev	Copies database devices from one managed SQL Server to another.	2-88
sdeldev	Deletes a database device from a managed SQL Server.	2-182
sgetdev	Gets information about database devices in a managed SQL Server.	2-228
ssetdev	Sets the configuration of a database device in a managed SQL Server.	2-325
scrtmirror	Creates a mirror for an existing device.	2-147
sdelmirror	Deletes a mirror from a device.	2-192

Dump Device Management

Table 2-9: Dump device management commands

Command	Synopsis	Page
scrtdumpdev	Creates a new dump device for backing up a database or transaction log.	2-138
scompdumpdev	Compares dump devices between managed SQL Servers and displays the differences.	2-42
scopydumpdev	Copies dump devices from one managed SQL Server to another.	2-91
sdeldumpdev	Deletes a dump device.	2-184
sgetdumpdev	Gets information about dump devices in a managed SQL Server.	2-233

Segment Management

Table 2-10: Segment management commands

Command	Synopsis	Page
scrtseg	Creates a segment in a database device.	2-162
scompseg	Compares segments between databases and displays the differences.	2-65
scopyseg	Copies segments from one database to another database.	2-110
sdelseg	Deletes a segment from a database.	2-200
sgetseg	Gets information about database segments and their relationships to other database entities.	2-266
ssetseg	Sets the attributes of a segment. Extends a segment to span additional existing database devices or removes devices from a segment.	2-351

Threshold Management

Table 2-11: Threshold management commands

Command	Synopsis	Page
scrtthresh	Creates a new threshold in a segment in a database in order to monitor space on the segment.	2-169
sdelthresh	Removes a threshold from a segment in a database.	2-205
ssetthresh	Sets the definition of a threshold in a segment in a database.	2-366

Profile Management

Table 2-12: Profile management commands

Command	Synopsis	Page
scrtprfmgr	Creates a profile manager for an associated SQL Server or database.	2-149
sgetprf	Gets information about names in a profile.	2-248
spopulateprf	Adds names to or removes names from a profile.	2-305
ssyncprf	Synchronizes the contents of a profile with its associated elements in SQL Server or in the database.	2-374

Cache Management

There are no Enterprise SQL Server Manager commands for managing cache. You must use the Enterprise SQL Server Manager Voyager to manage cache.

Policy Management

There are no Enterprise SQL Server Manager commands for managing policy. To implement policy from the command line, use the Tivoli commands **wgetpolm**, **wputpolm**, and **wlspolm**. To manage policy from the Enterprise SQL Server Manager GUI, use the SQL Server Profile Manager window or the Database Profile Manager window.

Database Object Management

ESSM includes commands for the following areas of database object management:

- Columns
- Datatypes
- Defaults
- Indexes
- Procedures
- Rules
- Tables
- Triggers
- Views

Column Management

Table 2-13: Column management commands

Command	Synopsis	Page
scrtcolumn	Creates a column in a table within a database.	2-162
ssetcolumn	Changes a datatype on a column in a table.	2-311

User Datatype Management

Table 2-14: User-defined datatype management commands

Command	Synopsis	Page
scrtdatatype	Creates a user-defined datatype within a database.	2-127
scompdatatype	Compares user-defined datatypes to datatypes in another database.	2-31
scopydatatype	Copies user-defined datatypes from one database to another.	2-81
sdeldatatype	Deletes user-defined datatypes from within a database.	2-178
sgetdatatype	Displays the definition of user-defined datatypes.	2-216
ssetdatatype	Changes bindings for a user-defined datatype in a database.	2-314

Default Management

Table 2-15: Default management commands

Command	Synopsis	Page
scrtdefault	Creates a default within a database. This default can also be bound to a column when it is created.	2-133
scompdefault	Compares default values to default values in other databases.	2-36
scopydefault	Copies default column values from one database to other.	2-86
sdeldefault	Removes the defaults from the database.	2-181
sgetdefault	Displays the defaults in a database.	2-226
ssetdefault	Changes the properties of a default.	2-323

Index Management

Table 2-16: Index management commands

Command	Synopsis	Page
scheckindex	Invokes **dbcc** to check the consistency of an index.	2-23
scrtindex	Creates an index on columns in a table.	2-162
scompindex	Compares indexes between two tables of the same name in different databases.	2-48
scopyindex	Copies index definitions to a table of the same name in another database.	2-96
sdelindex	Deletes indexes from a table.	2-188
sgetindex	Displays information about the indexes in a database table.	2-237
ssetindex	Changes the properties of an index.	2-329

Procedure Management

Table 2-17: Procedure management commands

Command	Synopsis	Page
scrtproc	Creates a stored procedure within a database.	2-151
scompproc	Compares stored procedures to stored procedures of the same name in another database.	2-55
scopyproc	Copies stored procedures from one database to another.	2-103
sdelproc	Deletes stored procedures from within a database.	2-194
sgetproc	Displays the definition of stored procedures.	2-251

Rule Management

Table 2-18: Rule management commands

Command	Synopsis	Page
scrtrule	Creates a rule in a database. This rule can also be bound to a column or user-defined datatype when it is created.	2-160
scomprule	Compares rules from different databases.	2-63
scopyrule	Copies rules from one database to another.	2-108
sdelrule	Deletes rules from a database.	2-199
sgetrule	Displays the rules.	2-266
ssetrule	Changes the properties of a rule.	2-351

Table Management

Table 2-19: Table management commands

Command	Synopsis	Page
schecktable	Invokes **dbcc** to check the consistency of a table.	2-26
scrttable	Creates a table within a database.	2-166
scomptable	Compares tables to tables of the same name in another database.	2-70
scopytable	Copies tables from one database to another.	2-113
sdeltable	Deletes tables from a database.	2-204
sgettable	Displays the definition of tables.	2-284
ssettable	Changes the properties of a table.	2-360

Trigger Management

Table 2-20: Trigger management commands

Command	Synopsis	Page
scrttrigger	Creates a trigger in a table.	2-171
scomptrigger	Compares triggers to triggers of the same name in another database.	2-72
scopytrigger	Copies triggers from one database to another. The triggers are copied to tables of the same name.	2-116
sdeltrigger	Deletes triggers from a database.	2-207
sgettrigger	Displays information about SQL Server triggers in a database.	2-290

View Management

Table 2-21: View management commands

Command	Synopsis	Page
scrtview	Creates a view within a database.	2-176
scompview	Compares views to views in another database.	2-77
scopyview	Copies views to other databases.	2-121
sdelview	Deletes views from within a database.	2-210
sgetview	Displays definitions of views.	2-294

Notices

Enterprise SQL Server Manager creates two notice groups in the Tivoli Management Environment:

- Sybase Administration
- Sybase Backup / Recovery

All Enterprise SQL Server Manager commands that create, delete, or modify SQL Server objects log a notice to the "Sybase Administration" notification group. Messages are in the following format:

```
Notice-id: ID_number
```

```
Date: Day/Date/Time/Year
Priority: Notice
Administrator: administrator_login
[action] [object]: [object path]
        Object Name : object_name
        Object Type : object_type
        Server : server_name
        [Database : db_name]
```

The priority of each of these messages is “Notice.”

Enterprise SQL Server Manager transaction rollback does not log notices.

All Enterprise SQL Server Manager commands that affect the backup and recovery of databases log a notice to the "Sybase Backup / Recovery" notice group. This includes the start and end of a backup, the start and end of a restore, and notification of use of the sp_volchanged stored procedure.

Notifications during profile management are rolled into one notification message.

Enterprise SQL Server Manager Commands Reference Manual Pages

The following pages in this chapter describe each command in detail. The commands are in alphabetical order.

schangevol

Function

Notifies the Backup Server that the operator performed the requested volume handling during a dump or load.

Syntax

```
schangevol –name db_name [–server server_name]
   -device device_name -session session_id
   {-abort | -retry | -proceed} [-filename file_name]
   [-volume volume_name] [-backupserver server_name]
   [-version] [–help]
```

Parameters

–name *db_name* – specifies the database being restored or backed up, where *db_name* is the name of the database.

–server *server_name* – specifies the SQL Server installation on which the database *db_name* resides, where *server_name* is the name of a managed SQL Server.

–device *device_name* – specifies the device on which the volume is mounted, where *device_name* is the pathname of the device. Use the *@devname* parameter specified in the Backup Server's volume change request.

–session *session_id* – identifies the Backup Server session that requested the volume change. Use the *@session_id* parameter that was specified in the Backup Server's volume change request.

–abort – stops the backup or restore operation.

–retry – starts the backup or restore operation again after you respond to an error message from SQL Server.

–proceed – continues the backup or restore operation after you change the volume.

–filename *file_name* – specifies the file that you want to load or back up.

–volume *volume_name* – specifies the volume name that appears in the ANSI tape label.

–backupserver *server_name* – identifies the Backup Server that requested the volume change if the Backup Server is not located on the same machine as SQL Server.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

```
Backup Server: 4.49.1.1: OPERATOR: Volume to be overwritten on
'/dev/rmt4' has not expired: creation date on this volume is
Sunday, Nov. 15, 1995, expiration date is Wednesday, Nov. 25,
1995.
Backup Server: 4.78.1.1: EXECUTE sp_volchanged
       @session_id = 8,
       @devname = '/auto/remote/pubs3/SERV/Masters/testdump',
       @action = { 'PROCEED' | 'RETRY' | 'ABORT' }
```

This message from the Backup Server indicates that a mounted tape's expiration date has not been reached. After changing tapes, the operator issues the command:

```
schangevol –name pubs2 -device /dev/rmt4
-session 8 -retry
```

This notifies the Backup Server session requesting a volume change (session ID 8) that the volume mounted on device */dev/rmt4* is now ready and to retry the backup on the new tape.

Permissions

To use **schangevol**, you must have the following roles:

TME	ESSM	SQL Server
any	**dump** or **load** or both	none

See Also

sdumpdb, sloaddb

scheckdb

Function

Checks the logical and physical consistency of a database and displays the results.

Syntax

```
scheckdb –name db_name
   [–overall [–skipindex]] | [–alloc [–fix]] |
   [-catalog]
   [–server server_name] [–version] [–help]
```

Parameters

–name *db_name* – specifies which database to check, where *db_name* is the name of the database.

–overall – checks the overall database consistency, including all indexes. This is the default option.

–skipindex – checks the overall database consistency, but skips checking non-clustered indexes on user tables. If you use this option, precede it with the **–overall** option.

–alloc – checks database allocations, but does not fix the allocation errors found.

–fix – checks database allocations and fixes the allocation errors found. This option puts the database (*db_name*) into single user mode. If you use this option, precede it with the **–alloc** option.

–catalog – checks the system catalog. Checks for consistency in and between system tables found in *db_path*.

–server *server_name* – specifies the SQL Server installation on which the database *db_name* resides, where *server_name* is the name of a managed SQL Server. The default is the current managed SQL Server collection (see “Command Context”).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`scheckdb –name sampledb –server BOSTON`**

   Checks the overall database consistency for the database named *sampledb* on the managed SQL Server BOSTON:

   ```
   Checking sampledb
   Checking 1
   The total number of data pages in this table is 3.
   Table has 52 data rows.
   Checking 2
   The total number of data pages in this table is 7.
   Table has 286 data rows.
   Checking 3
   The total number of data pages in this table is
   190.
   *** NOTICE:  Notification of log space used/free
   cannot be reported because the log segment is not
   on its own device.
   Table has 4681 data rows.
   DBCC execution completed. If DBCC printed error
   messages, contact a user with System Administrator
   (SA) role.
   ```

2. **`scheckdb –name master –catalog`**

   Checks the system catalog for the *master* database in the current SQL Server collection:

   ```
   Checking master
   The following segments have been defined for
   database 1 (database name master).
   virtual start addr      size      segments
   --------------------    ------    -----------------
   4                       1536
                                         0
                                         1
                                         2
   5636                     2560
                                         2
   DBCC execution completed. If DBCC printed error
   messages, contact a user with System Administrator
   (SA) role.
   ```

3. **`scheckdb –name pubs2 –alloc`**

   Checks the database allocations and system catalog for the *pubs2* database in the current SQL Server collection:

```
Checking pubs2
Database ‘pubs2’ is not in single user mode – may find spurious
allocation problems due to transactions in progress.
***************************************************
TABLE: authors          OBJID = 16003088
INDID=1  FIRST=489       ROOT=313        SORT=1
        Data level: 1.  1 Data  Pages in 1 extents.
        Indid     : 1.  1 Index Pages in 1 extents.
INDID=2  FIRST=145       ROOT=145        SORT=1
        Indid     : 2.  1 Index Pages in 1 extents.
TOTAL # of extents = 3
***************************************************
TABLE: publishers               OBJID = 48003202
INDID=1  FIRST=481       ROOT=473        SORT=1
        Data level: 1.  1 Data  Pages in 1 extents.
        Indid     : 1.  1 Index Pages in 1 extents.
TOTAL # of extents = 2
***************************************************
Processed 51 entries in the sysindexes for dbid 5.
Alloc page 0 (# of extent=32 used pages=90 ref pages=82)
Alloc page 256 (# of extent=32 used pages=103 ref pages=93)
Alloc page 512 (# of extent=32 used pages=199 ref pages=199)
Alloc page 768 (# of extent=19 used pages=139 ref pages=138)
Total (# of extent=115 used pages=531 ref pages=512) in this
database
DBCC execution completed. If DBCC printed error messages,
contact a user with System Administrator (SA) role.
```

(Example output for only a portion of the database is shown here.)

Comments

- Before using **scheckdb**, you must know the name of the database to check.
- The Database Owner has implicit permission to use **scheckdb**.
- **scheckdb** returns results in the form of an exception. Completion status is always “-1”.

Permissions

To use **scheckdb**, you must have the following roles:

TME	ESSM	SQL Server
any	**space**	**sa_role** or Database Owner

See Also

schecktable, scheckindex, scompdb, scopydb, scrtdb, sdeldb, sgetdb, ssetdb

scheckindex

Function

Invokes dbcc to check the consistency of an index.

Syntax

```
scheckindex -name table_name.index_name [-fix]
   [-report {optimized | full | fast}]
   [-database db_path] [-version] [-help]
```

Parameters

–name *table_name.index_name* – specifies the name of the index to check. The index name must be prefixed with the table name.

–fix – fixes problems encountered. If you are running **–fix** on a system table, the database must be in single user mode. The default is to generate a report and not to fix errors.

–report – specifies the type of report to run. You must specify one of the following values:

- **optimized** reports allocation pages listed in the Object Allocation Map pages for the table.
- **full** reports all types of allocation errors.
- **fast** reports pages that are not allocated in the extent.

The default report setting is **optimized**.

–database *db_path* – specifies the database in which to create the type. The default is the current managed database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

➤ ***Note***

The output of **scheckindex** is reported to *stderr* in the form of an error message.

Example

```
scheckindex -name sysobjects.objects
-database PARIS/sybsystemprocs -fix
```

This command invokes **dbcc** to check the index sysobjects in the table *sysobjects*, owned by Database Owner, in the database *sybsystemprocs* in the managed SQL Server PARIS. It checks that index and data pages are correctly allocated, that no page is allocated that is not used, and that no page is used that is not allocated. This command fixes problems that it finds and prints a full report.

```
12/06/95 10:39:50: dbcc results

->    12/06/95 10:39:50:
***************************************************

->    12/06/95 10:39:50: TABLE: sysobjectsOBJID = 1

->    12/06/95 10:39:50: INDID=1 FIRST=1 ROOT=8 SORT=0->12/06/95
10:39:50: Data level: 1.  8 Data  Pages in 2 extents.

->    12/06/95 10:39:50: Indid  : 1.  1 Index Pages in 1 extents.

->    12/06/95 10:39:50: TOTAL # of extents = 3

->    12/06/95 10:39:50: Alloc page 0 (# of extent=1 used pages=2

      ref pages=2)

->    12/06/95 10:39:50: Alloc page 0 (# of extent=1 used pages=8

      ref pages=8)

->    12/06/95 10:39:50: Alloc page 2560 (# of extent=1
used pages=2   ref pages=2)

->    12/06/95 10:39:50: Total (# of extent=3 used pages=12
ref pages=12) in this database

->    12/06/95 10:39:50: DBCC execution completed. If DBCC
printed error messages, contact a user with System Administrator
(SA) role.
```

Comment

scheckindex returns results in the form of an exception. Completion status is always “-1”.

Permissions

To use **scheckindex**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	Database Owner

See Also

scheckdb, schecktable, sgetindex, scrtindex, sdelindex, ssetindex, scopyindex, scompindex

schecktable

Function

Invokes **dbcc** to check the consistency of a table.

Syntax

```
schecktable -name table_name
   [-overall [-skipindex]] |
   [-alloc [-fix] [-report{optimized | full | fast}]]|
   [-reindex] | [-text]
   [-database db_path] [-version] [-help]
```

Parameters

–name *table_name* – specifies the name of the table to check. The name of the table can be prefixed with the name of the owner as follows: *owner.table_name*.

–overall – checks that the index and data pages are correctly linked, indexes are properly sorted, all pointers are consistent, and the data rows on each page have entries in the first page of an allocation map. This is the default option.

–skipindex – specified with **–overall**, this parameter causes **schecktable** to skip the checking of non-clustered indexes on user tables.

–alloc – checks that index and data pages are correctly allocated, that no page is allocated that is not used, and that no page is used that is not allocated.

–fix – fixes problems encountered. **–fix** can only be specified with the **–alloc** option. If you are running **–fix** on a system table, the database must be in single user mode. The default is to generate a report and not to fix errors.

–report – specifies the type of report to run. You must specify one of the following values:

- **optimized** reports allocation pages listed in the Object Allocation Map pages for the table.
- **full** reports all types of allocation errors.
- **fast** reports pages that are not allocated in the extent.

–report can only be specified if the **–alloc** option is specified. The default report setting is **“optimized”**.

–reindex – checks the integrity of indexes on user tables. It drops and rebuilds indexes it suspects are corrupt.

–text – upgrades text values after a SQL Server character set has been changed to a multibyte character set.

–database *db_path* – specifies the database in which to create the type. The default is the current managed database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

➤ ***Note***

The output of **schecktable** is reported to *stderr* in the form of an error message.

Example

```
schecktable -database PARIS_DEV/pubs2_test -name
dbo.titles -overall
```

This command runs **dbcc** against the table *titles*, owned by *dbo*, in the database *pubs2* in the managed SQL Server PARIS_DEV. It checks that the index and data pages are correctly linked, indexes are properly sorted, all pointers are consistent, and the data rows on each page have entries in the first page of an allocation map. It does not check non-clustered indexes in user tables.

```
12/06/95 10:39:50: dbcc results
->  12/06/95 10:39:50: Checking dbo.titles
->  12/06/95 10:39:50: The total number of data
pages in this table is 3.
->  12/06/95 10:39:50: Table has 18 data rows.
->  12/06/95 10:39:50: DBCC execution completed.
If DBCC printed error messages, contact a user
with System Administrator (SA) role.
```

Comment

schecktable returns results in the form of an exception. Completion status is always “-1”.

Permissions

To use **schecktable**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	Database Owner. For the **–text** and **–reindex** options, you must also be the table owner.

See Also

scheckdb, scheckindex, sgettable, scrttable, sdeltable, ssettable, scopytable, scomptable

scompcmdperm

Function

Compares command permissions between two databases and displays the differences. The **scompcmdperm** command compares permissions on the following commands:

- **create database**
- **create default**
- **create procedure**
- **create rule**
- **create table**
- **create view**
- **dump database**
- **dump transaction**

Syntax

```
scompcmdperm
   [–names users_or_groups_or_role_names [–wildcard]]
   {[–source db_path] [–target db_path]}
   [–version] [–help]
```

Parameters

–names *users_or_groups_or_role_names* – specifies the user, group, or role whose command permissions you want to compare.

–wildcard – enables wildcard matching on *users_or_groups_or_role_names*. (see “Wildcards” on page 1-4).

–source *db_path* – specifies the database to compare, where *db_path* can be any valid managed database name (see “Database Names” on page 1-16). The default is the current database collection (see “Command Context” on page 1-15).

–target *db_path* – specifies the new database name, where *db_path* can be any valid managed database name (see “Database Names” on page 1-16). You must specify a target database. The target database must not already exist before the copy operation (see “Comments”).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`scompcmdperm -s KOKO/pubs2 -t KOKO/sales`**

   Compares the command permissions on two databases, *pubs2* and *sales* on the managed Server KOKO.

```
<<< Source: KOKO/pubs2
>>> Target: KOKO/sales
---------------------------------------------------
---------- Grant Create Default guest dbo ---------
---------- Grant Create Procedure public dbo ------
---------- Grant Create Rule guest dbo ----------
---------- Grant Create Table guest dbo ----------
---------- Grant Create View guest dbo ----------
```

2. **`scompcmdperm -names bwebster -s KOKO/pubs2 -t AGRA/pubs2`**

   Compares the command permissions for user *bwebster* in two different databases on two different managed Servers: *pubs2* on KOKO and *pubs2* on AGRA. No differences are found.

```
<<< Source: KOKO/pubs2
>>> Target: AGRA/pubs2
---------------------------------------------------
No differences
```

Comments

Before using **scompcmdperm**, you need the following information:

- The name of the databases containing the command permissions to be compared.
- The names of any users, groups, or roles whose command permissions you want to compare.

Permissions

To use **scompcmdperm**, you must have the following roles:

TME	ESSM	SQL Server
any	any	**sso_role**

See Also

scompobjperm, sgetcmdperm, sgetobjperm

scompdatatype

Function

Compares user datatypes in a database with datatypes in another database.

Syntax

```
scompdatatype [-names datatype_names] [-bindings]
   [-source db_path] [-target db_path]
   [-wildcard] [-version] [-help]
```

Parameters

–names *datatype_names* – specifies the names of the datatypes to compare. The names of the datatypes can be prefixed with the name of the owner as follows: owner.datatype_name. If you do not specify any names, all the datatypes defined in the database are compared. The following information will be compared:

- physical_type
- length
- precision
- scale
- identity / null / nonull setting

–bindings – causes the datatype's bindings, the bound rule name and bound default name, to be compared.

–source *db_path* – specifies the database from which to compare the datatypes, where *db_path* can be any valid managed database name. The default source is the current managed database. If you do not specify a source database, you must specify a target database using **–target**.

–target *db_path* – specifies the database in which to compare the datatypes from the source database, where *db_path* can be any valid managed database name. The default target is the current managed database. If you do not specify a target database, you must specify a source database using -source.

–wildcard – enables wildcard pattern matching on the datatype names.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scompdatatype -name ssn -source BOSTON/pubs2
-target PARIS/pubs2
```

This command compares the user datatype *ssn* in the database *pubs2*, in the managed SQL Server BOSTON, with the same user datatype in the database *pubs2*, in the managed SQL Server PARIS.

```
---------- ssn ----------
Length:
< 11
> 12
Nulls List:
< NoNulls
> Nulls
```

Permissions

To use **scompdatatype**, you must have the following roles:

TME	ESSM	SQL Server
any	any	any

See Also

sgetdatatype, scrtdatatype, sdeldatatype, ssetdatatype, scopydatatype

scompdb

Function

Compares space allocation, devices, and database options between databases and displays the differences. **scompdb** compares database attributes that you define using **scrtdb** and **ssetdb**.

Syntax

```
scompdb {[–source db_path] [–target db_path]}
   [–version] [–help]
```

Parameters

–source *db_path* – specifies the database with which to compare the target database, where *db_path* can be any valid managed database name (see "Database Names" on page 1-16). The default is the current database collection (see "Command Context" on page 1-15). If you do not specify a source database, you must specify a target database using **–target** (see "Comments"). Output lines beginning with "<" refer to the source database.

–target *db_path* – specifies the database with which to compare the source database, where *db_path* can be any valid managed database name (see "Database Names" on page 1-16). The default is the current database collection (see "Command Context" on page 1-15). If you do not specify a target database, you must specify a source database using **–source** (see "Comments"). Output lines beginning with ">" refer to the target database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scompdb –s KOKO/pubs2 –t KOKO/sales
```

Compares the database *pubs2* with the *sales* database in the KOKO managed SQL Server and displays the differences:

```
<<< Source: KOKO/pubs2
>>> Target: KOKO/sales
---------------------------------------------------
Database Owner:
< sa
> sybase
Database Creation Date:
< Sep 21 1994 11:48AM
> Nov 22 1994  2:28PM
Dump Transaction Date:
< Sep 21 1994 11:49AM
> Nov 22 1994  2:28PM
Number of Users:
< 3
> 2
Database Configurations:
Space Utilization:
Reserved in MB:
< 1.07227
> 0.652344
Data in MB:
< 0.162109
> 0.0585938
Index Size in MB:
< 0.0917969
> 0.046875
Reserved but Unused in MB:
< 0.818359
> 0.546875
Device Information:
Device Fragment:
< data_dev1
> test_dev1
Free kbytes:
< 912
> 1360
```

Comments

- Before using **scompdb**, you must know the names of the databases to compare.
- The **–source** and **–target** database names must be different. If you specify the same database name for both options, **scompdb** displays an error message.

  Because both **–source** and **–target** options default to the current database collection (see “Command Context” on page 1-15), you must supply at least one of the two database names. If you omit both options, **scompdb** displays an error message.
- **scompdb** does not compare database objects.
- You can compare databases that have the same name if they are on different managed SQL Servers.

Permissions

To use **scompdb**, you must have the following roles:

TME	ESSM	SQL Server
any	any	**sa_role** or valid database user

See Also

scheckdb, scompgroup, scompuser, scopydb, scrtdb, sdeldb, sgetdb, ssetdb

scompdefault

Function

Compares default values in a database with default values in other databases.

Syntax

```
scompdefault [-names default_names] [-bindings]
   [-source db_path] [-target db_path] [-wildcard]
   [-version] [-help]
```

Parameters

–names *default_names* – specifies the names of the defaults to compare. The names of the defaults can be prefixed with the name of the owner as follows: *owner.default_name*. If you do not specify **–names**, all the defaults defined in the database are compared. The name and value is compared.

–bindings – specifies comparison of any bindings to user data types and columns.

–source *db_path* – specifies the database from which to compare the defaults, where *db_path* can be any valid managed database name. The default source is the current managed database. If you do not specify a source database, you must specify a target database using **–target**.

–target *db_path* – specifies the database in which to compare the defaults from the source database, where *db_path* can be any valid managed database name. The default target is the current managed database. If you do not specify a target database, you must specify a source database using **–source**.

–wildcard – enables wildcard pattern matching on the default names.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scompdefault -names sex_def
-source PARIS/devel_hrdb
-target ROME/prod_hrdb
```

This command compares the value of the default *sex_def* in the database *devel_hrdb* in the managed SQL Server PARIS, with the value for that default in database *prod_hrdb* in the managed SQL Server ROME.

```
<<< Source: PARIS/devel_hrdb
>>> Target: ROME/prod_hrdb
--------------------------------
---------- dbo.sex_def ---------
Value
<"M"
>"F"
```

Permissions

To use **scompdefault**, you must have the following roles:

TME	ESSM	SQL Server
any	any	any

See Also

sgetdefault, scrtdefault, sdeldefault, ssetdefault, scopydefault

scompdev

Function

Compares database devices between managed SQL Servers and displays the differences. **scompdev** compares the following device attributes:

- Logical device name
- Physical device name
- Size
- Default pool
- Mirror device name
- Mirror status
- Starting virtual address
- Disk controller
- Virtual device number

Syntax

```
scompdev [–names device_names [–wildcard]]
   {[–source server_name] [–target server_name]}
   [–version] [–help]
```

Parameters

–names *device_names* – specifies which database devices to compare, where *device_names* lists one or more database device names. **scompdev** compares each named device on the source SQL Server with the corresponding name on the target SQL Server. Without this option, **scompdev** compares all database devices occurring on both SQL Servers. If the list of device names contains spaces, you must enclose the list in double quotation marks (see “Option Lists” on page 1-3).

–wildcard – enables wildcard matching on *device_names* (see “Wildcards” on page 1-4).

–source *server_name* – specifies the managed SQL Server with which to compare *device_names* on the target SQL Server, where *server_name* can be any valid managed SQL Server name (see “SQL Server Names” on page 1-15). The default is the current managed SQL Server collection (see “Command Context” on page 1-15). If you do not specify a source SQL Server, you must specify a target SQL Server using **–target** (see “Comments”). Output lines beginning with “<” refer to the source SQL Server.

–target *server_name* – specifies the managed SQL Server with which to compare *device_names* on the source SQL Server, where *server_name* can be any valid managed SQL Server name (see “SQL Server Names” on page 1-15). The default is the current managed SQL Server collection (see “Command Context” on page 1-15). If you do not specify a target SQL Server, you must specify a source SQL Server using **–source** (see “Comments”). Output lines beginning with “>” refer to the target SQL Server.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scompdev –s KOKO –t AGRA
```

Compares all database devices on the managed SQL Server KOKO with those on the managed SQL Server AGRA and reports the differences:

```
<<< Source: KOKO
>>> Target: AGRA
---------------------------------------------------
---------- data_dev1 ----------
Physical Name:
< /pinehome/system1001/KOKO_data_dev1.dat
> /pinehome/system1001/AGRA_data_dev1.dat
---------- log_dev1 ----------
Physical Name:
< /pinehome/system1001/KOKO_log_dev1.dat
> /pinehome/system1001/AGRA_log_dev1.dat
---------- sybsecurity ----------
Physical Name:
< /pinehome/system1001/KOKO_AUDIT.dat
> /pinehome/system1001/AGRA_AUDIT.dat
---------- sysprocsdev ----------
Physical Name:
< /pinehome/system1001/KOKO_PROCS.dat
> /pinehome/system1001/AGRA_PROCS.dat
---------- test_dev1 ----------
Physical Name:
< /pinehome/system1001/KOKO_test_dev1.dat
> /pinehome/system1001/AGRA_test_dev1.dat
```

```
Reserved Space in Megabytes:
< 2
> 0
Unused Space in Megabytes:
< 1.328
> 0
---------- test_dev2 ----------
Physical Name:
> /pinehome/system1001/AGRA_test_dev2.dat
Size in Megabytes:
> 2
Virtual Device Number:
> 6
Mirror Name:
> null
Status:
> 2
Virtual Start:
> 0
Controller Type:
> 0
Reserved Space in Megabytes:
> 0
Unused Space in Megabytes:
> 0
```

Comments

- Before using **scompdev**, you need the following information:
  - The database device names to compare
  - The managed SQL Servers on which the devices reside
- The **–source** and **–target** SQL Server names must be different. If you specify the same server name for both options, **scompdev** displays an error message.

  Because both **–source** and **–target** options default to the current managed SQL Server collection (see "Command Context" on page 1-15), you must supply at least one of the two SQL Server names. If you omit both options, **scompdev** displays an error message.

Permissions

To use **scompdev**, you must have the following roles:

TME	ESSM	SQL Server
any	any	none

See Also

scompdb, scopydev, scrtdev, sdeldev, sgetdev, ssetdev

scompdumpdev

Function

Compares dump devices between managed SQL Servers and displays the differences. **scompdumpdev** compares the following attributes:

- Logical device name
- Physical device name
- Disk / tape status
- Size

Syntax

```
scompdumpdev [-names device_names [-wildcard]]
   {[-source server_name] [-target server_name]}
   [-version] [-help]
```

Parameters

–names *device_names* – specifies which dump devices to compare, where *device_names* lists one or more dump device names. **scompdumpdev** compares each dump device on the source SQL Server or the target SQL Server whose name is in the list of device names. Without this option, **scompdumpdev** compares all dump devices occurring on either SQL Server. If the list of device names contains spaces, you must enclose the list in double quotation marks.

–wildcard – enables wildcard matching on *device_names* (see “Wildcards” on page 1-4).

–source *server_name* – specifies the managed SQL Server with which to compare *device_names* on the target SQL Server, where *server_name* can be any valid managed SQL Server name (see “SQL Server Names” on page 1-15). The default is the current managed SQL Server collection (see “Command Context” on page 1-15). If you do not specify a source SQL Server, you must specify a target SQL Server using **–target** (see “Comments”). Output lines beginning with “<” refer to the source SQL Server.

–target *server_name* – specifies the managed SQL Server with which to compare *device_names* on the source SQL Server, where *server_name* can be any valid managed SQL Server name (see

"SQL Server Names" on page 1-15). The default is the current managed SQL Server collection (see "Command Context" on page 1-15). If you do not specify a target SQL Server, you must specify a source SQL Server using **–source** (see "Comments"). Output lines beginning with ">" refer to the target SQL Server.

–version – prints the release and copyright of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`scompdumpdev -source KOKO -target AGRA`**

   Compares the dump devices on two managed Servers, KOKO and AGRA. The dump devices are the same.

   ```
   <<< Source: KOKO
   >>> Target: AGRA
   ---------------------------------------------------
   No differences
   ```

2. **`scompdumpdev -source KOKO -target CHORD`**

   Compares the dump devices on two managed Servers, KOKO and CHORD and displays the differences.

   ```
   <<< Source: KOKO
   >>> Target: CHORD
   ---------------------------------------------------
   ---------- tapedump2 ----------
   Physical Name:
   > /dev/rst0
   Controller Type:
   > 2
   Size in Megabytes:
   > 625
   ```

Comments

- Before using **scompdumpdev**, you must have the following information:
  - The names of specific dump devices you want to compare
  - The name of the source or target managed SQL Server associated with the devices
- The **–source** and **–target** SQL Server names must be different. If you specify the same server name for both options, **scompdumpdev** displays an error message.

  Because both **–source** and **–target** options default to the current managed SQL Server collection (see "Command Context" on page 1-15), you must supply at least one of the two SQL Server names. If you omit both options, **scompdumpdev** displays an error message.

Permissions

To use **scompdumpdev**, you must have the following roles:

TME	ESSM	SQL Server
any	any	none

See Also

scopydumpdev, scrtdumpdev, sdeldumpdev, sgetdumpdev, ssetdumpdev

scompgroup

Function

Compares groups between databases and displays the differences. The scompgroup command compares user membership in each group.

Syntax

```
scompgroup [–names group_names [–wildcard]]
   {[–source db_path] [–target db_path]}
   [–version] [–help]
```

Parameters

–names *group_names* – specifies which group names to compare. *group_names* lists one or more group names. **scompgroup** compares each named group on the source database with the corresponding name on the target database. Without this option, **scompgroup** compares all groups occurring on both databases. If the list of group names contains spaces, you must enclose the list in double quotation marks (see "Option Lists" on page 1-3).

–wildcard – enables wildcard matching on *group_names* (see "Wildcards" on page 1-4).

–source *db_path* – specifies the database with which to compare *group_names* on the target database where *db_path* can be any valid managed database name (see "Database Names" on page 1-16). The default is the current database collection (see "Command Context" on page 1-15). If you do not specify a source database, you must specify a target database using **–target** (see "Comments"). Output lines beginning with "<" refer to the source database.

–target *db_path* – specifies the database with which to compare *group_names* on the source database, where *db_path* can be any valid managed database name (see "Database Names" on page 1-16). The default is the current database collection (see "Command Context" on page 1-15). If you do not specify a target database, you must specify a source database using **–source** (see "Comments"). Output lines beginning with ">" refer to the target database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`scompgroup –source KOKO/pubs2 –target KOKO/sales`**

   Compares all groups in the *pubs2* database with those in the *sales* database on the managed SQL Server KOKO and displays the differences:

   ```
   <<< Source: KOKO/pubs2
   >>> Target: KOKO/sales
   ---------------------------------------------------
   ---------- bucks ----------
   Users:
   User:
   > bennyg
   User:
   > busyb
   ---------- jazz ----------
   Users:
   User:
   < bennyg
   User:
   < webster
   ---------- public ----------
   Users:
   User:
   < guest
   ```

2. **`scompgroup -source KOKO/pubs2 -target AGRA/pubs2`**

   Compares the groups in the *pubs2* database on managed Server KOKO with the groups in the *pubs2* database on managed Server AGRA.

   ```
   <<< Source: KOKO/pubs2
   >>> Target: AGRA/pubs2
   ---------------------------------------------------
   ---------- jazz ----------
   Users:
   User:
   < bennyg
   User:
   < webster
   ```

Comments

- Before using **scompgroup**, you need the following information:
  - The names of the groups to compare (unless you want to compare all groups)
  - The name of the databases in which the groups exist
- The **–source** and **–target** database names must be different. If you specify the same database name for both options, **scompgroup** displays an error message.

  Because both **–source** and **–target** options default to the current database collection (see “Command Context” on page 1-15), you must supply at least one of the two database names. If you omit both options, **scompgroup** displays an error message.
- **scompgroup** does not compare group permissions.

Permissions

To use **scompgroup**, you must have the following roles:

TME	ESSM	SQL Server
any	any	**sso_role** or valid database user

See Also

scompuser, scopygroup, scrtgroup, sdelgroup, sgetgroup, ssetgroup

scompindex

Function

Compares indexes between two tables of the same name in different databases.

Syntax

```
scompindex [-names index_names] [-source db_path]
   [-target db_path] [-wildcard] [-version] [-help]
```

Parameters

–names *index_names* – specifies the names of the indexes to compare in the format owner.table.indexname. If you do not use **–names**, all indexes defined in the default or specified database are compared. Any of the three parts of *index_names* can be replaced with a wildcard. The owner and table name are optional. The following information is compared:

- duplicate key setting
- duplicate row setting
- unique setting
- segment
- columns

–source *db_path* – specifies the database from which to compare the indexes, where *db_path* can be any valid managed database name. The default source is the current managed database. If you do not specify a source database, you must specify a target database using **–target**.

–target *db_path* – specifies the database in which to compare the indexes from the source database, where *db_path* can be any valid managed database name. The default target is the current managed database. If you do not specify a target database, you must specify a source database using **–source**.

–wildcard – enables wildcard pattern matching on the index names.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scompindex -source PARIS/pubs2
-target BOSTON/pubs2 -name sales.salesind
```

This command compares the index *salesind* in the database *pubs2*, in the managed SQL Server PARIS, with the table of the same name in the database *pubs2* in the managed SQL Server BOSTON.

```
---------- salesind dbo sales ----------
Unique:
< True
> False
Clustered:
< True
> False
Index Columns:
Column Name:
< ord_num
> stor_id
Column Name:
< stor_id
Index Segments:
Segment Name:
< dataseg_2
> default
```

Comment

You can omit the first two parts (owner and table name) of the *index_names* argument. The name "a.b" implies that the owner was omitted. The name "a" implies that both the owner and table were omitted. If you specify just the index name, all indexes with that name are retrieved regardless of the owner or the table on which they are defined.

Permissions

To use **scompindex**, you must have the following roles:

TME	ESSM	SQL Server
any	any	any

See Also

scheckindex, sgetindex, scrtindex, sdelindex, ssetindex, scopyindex

scomplogin

Function

Compares logins between managed SQL Servers and displays the differences. The **scomplogin** command compares the following login attributes:

- Full name
- Database
- Password date and expiration
- Connection status
- Server user ID (suid)
- Language
- Lock status
- Roles

Syntax

```
scomplogin [-names login_names [-wildcard]]
   {[-source server_name] [-target server_name]}
   [-version] [-help]
```

Parameters

–names *login_names* – specifies which login names to compare, where *login_names* lists one or more login names. **scomplogin** compares each named login on the source SQL Server with the corresponding name on the target SQL Server. Without this option, **scomplogin** compares all logins occurring on both SQL Servers. If the list of login names contains spaces, you must enclose the list in double quotation marks (see “Option Lists” on page 1-3).

–wildcard – enables wildcard matching on *login_names* (see “Wildcards” on page 1-4).

–source *server_name* – specifies the managed SQL Server with which to compare *login_names* on the target SQL Server, where *server_name* can be any valid managed SQL Server name (see “SQL Server Names” on page 1-15). The default is the current managed SQL Server collection (see “Command Context” on page 1-15). If you do not specify a source SQL Server, you must specify a target SQL

Server using **–target** (see “Comments”). Output lines beginning with “<” refer to the source SQL Server.

–target *server_name* – specifies the managed SQL Server with which to compare *login_names* on the source SQL Server, where *server_name* can be any valid managed SQL Server name (see “SQL Server Names” on page 1-15). The default is the current managed SQL Server collection (see “Command Context” on page 1-15). If you do not specify a target SQL Server, you must specify a source SQL Server using **–source** (see “Comments”). Output lines beginning with “>” refer to the target SQL Server.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scomplogin -names "harryc probe" -source KOKO -
target AGRA
```

Compares the logins *harryc* and *probe* on two managed Servers, KOKO and AGRA.

```
<<< Source: KOKO
>>> Target: AGRA
---------------------------------------------------
Password Date:
< 09/21/94 11:49:25
> 09/21/94 11:49:17
Server User ID:
< 5
> 4
Password Date:
< 09/21/94 11:40:44
> 09/21/94 11:34:50
```

Comments

- Before using **scomplogin**, you need the following information:
  - The login names to compare
  - The name of the source or target managed SQL Server
- The **–source** and **–target** SQL Server names must be different. If you specify the same server name for both options, **scomplogin** displays an error message.

Because both **–source** and **–target** options default to the current managed SQL Server collection (see “Command Context” on page 1-15), you must supply at least one of the two SQL Server names. If you omit both options, **scomplogin** displays an error message.

- **scomplogin** compares password dates and expirations, but does not compare the passwords themselves.
- **scomplogin** does not compare database users associated with a SQL Server login. To compare database users, use **scompuser**.

Permissions

To use **scomplogin**, you must have the following roles:

TME	ESSM	SQL Server
any	any	**sa_role**

See Also

scopylogin, scrtlogin, sdellogin, sgetlogin, ssetlogin

scompobjperm

Function

Compares object permissions for specified users, groups, or roles between databases, and displays the differences. The **scompobjperm** command compares permissions on:

- tables
- views
- procedures

Syntax

```
scompobjperm [–names object_names [–wildcard]]
   {[–source db_path] [–target db_path]}
   [–version] [–help]
```

Parameters

–names *object_names* – specifies the users, groups, roles, tables or procedures that have object permissions you want to compare.

–wildcard – enables wildcard matching on *users_or_groups_or_role_names* (see “Wildcards” on page 1-4).

–source *db_path* – specifies the database to compare, where *db_path* can be any valid managed database name (see “Database Names” on page 1-16). The default is the current database collection (see “Command Context” on page 1-15).

–target *db_path* – specifies the new database name, where *db_path* can be any valid managed database name (see “Database Names” on page 1-16). You must specify a target database. The target database must not already exist before the copy operation (see “Comments”).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scompobjperm -name discounts -source KOKO/pubs2 -
target AGRA/pubs2
```

```
<<< Source: KOKO/pubs2
>>> Target: AGRA/pubs2
---------------------------------------------------
---------- dbo.discounts U  References barneyb
Grantable:
< NO
Grantor Name:
< dbo
Permission Type:
< Grant
---------- dbo.discounts U  Update barneyb
Grantable:
< YES
Grantor Name:
< dbo
Permission Type:
< With Grant
```

Compares the object permissions of user *barneyb* on the *KOKO* / *pubs2* and *AGRA/pubs2* databases. The heading for each segment of the display is in the format:

```
object_name type type_of_permission user/group
```

Comment

Before using **scompobjperm**, you need the following information:

- The users, groups, or roles whose object permissions you want to compare.
- The databases where the users, groups, or roles are located.

Permissions

To use **scompobjperm**, you must have the following roles:

TME	ESSM	SQL Server
any	any	**sso_role**

See Also

scompcmdperm, scompuser, sgetcmdperm, sgetgroup, sgetobjperm, sgetuser

scompproc

Function

Compares stored procedures with stored procedures of the same name in another database.

Syntax

```
scompproc [-names procedure_names]
   [-source db_path] [-target db_path]
   [-wildcard] [-version] [-help]
```

Parameters

–names *procedure_names* – specifies the names of the stored procedures to compare. The names of the stored procedures can be prefixed with the name of the owner, as follows: *owner.procedure_name*. If you do not specify **–names**, all the stored procedures defined in the database are compared. The following information is compared: stored procedure schema including name, parameters, recompile status, and SQL.

–source *db_path* – specifies the database from which to compare the stored procedures, where *db_path* can be any valid managed database name. The default source is the current managed database. If you do not specify a source database, you must specify a target database using **–target**.

–target *db_path* – specifies the database in which to compare the stored procedures from the source database, where *db_path* can be any valid managed database name. The default target is the current managed database. If you do not specify a target database, you must specify a source database using **–source**.

–wildcard – enables wildcard pattern matching on the stored procedure names.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

➤ ***Note***

Compares of large SQL text may be very cumbersome.

Example

```
scompproc -name storeid_proc -source BOSTON/pubs2
-target PARIS/pubs2
```

This command compares the stored procedure *storeid* in the database *pubs2*, in the managed SQL Server BOSTON, with the same stored procedure in the database *pubs2*, in the managed SQL Server PARIS.

```
---------- storeid_proc dbo ----------
Procedure Create Date:
< Oct 31 1995  9:45AM
> Oct 23 1995 11:49AM
Summary Text:
Procedure text:
<
create proc storeid_proc
@stor_id        char(4)
as
select stor_name,
stor_id
from stores
where stor_id = @stor_id
return @@rowcount

>
create proc storeid_proc
@stor_id        char(4)
as
select stor_name,
    stor_id,
    stor_address,
    city,
    state,
    postalcode,
    country
from stores
where stor_id = @stor_id
return @@rowcount
```

Permissions

To use **scompproc**, you must have the following roles:

TME	ESSM	SQL Server
any	any	any

See Also

sgetproc, scrtproc, sdelproc, scopyproc

scomprmtserver

Function

Compares remote SQL Servers between managed SQL Servers and displays the differences. The **scomprmtserver** command compares all remote server attributes, including:

- Timeout
- Network password encryption
- Local login mapping configurations

Optionally, **scomprmtserver** also compares the following remote login attributes:

- Remote login name
- Local login name
- Trusted password mode configuration

Syntax

```
scomprmtserver
   [–names remote_server_names [–wildcard]]
   [–remotelogins]
   {[–source server_name] [–target server_name]}
   [–version] [–help]
```

Parameters

–names *remote_server_names* – specifies which remote SQL Servers to compare, where *remote_server_names* lists one or more remote server names. **scomprmtserver** compares each named remote SQL Server accessible with the source SQL Server with the corresponding name on the target SQL Server. Without this option, **scomprmtserver** compares all remote SQL Servers accessible to either SQL Server. Separate the remote SQL Server names with a space and enclose the list in double quotation marks (see “Option Lists” on page 1-3).

–wildcard – enables wildcard matching on *remote_server_names* (see “Wildcards” on page 1-4).

–remotelogins – compares the remote logins associated with each remote SQL Server, in addition to the other remote SQL Server attributes being compared.

–source *server_name* – specifies the managed SQL Server with which to compare *remote_server_names* on the target SQL Server, where *server_name* can be any valid managed SQL Server name (see "SQL Server Names" on page 1-15). The default is the current managed SQL Server collection (see "Command Context" on page 1-15). If you do not specify a source SQL Server, you must specify a target SQL Server using **–target** (see "Comments"). Output lines beginning with "<" refer to the source SQL Server.

–target *server_name* – specifies the managed SQL Server with which to compare *remote_server_names* on the source SQL Server, where *server_name* can be any valid managed SQL Server name (see "SQL Server Names" on page 1-15). The default is the current managed SQL Server collection (see "Command Context" on page 1-15). If you do not specify a target SQL Server, you must specify a source SQL Server using **–source** (see "Comments"). Output lines beginning with ">" refer to the target SQL Server.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`scomprmtserver –names "ENGLAND FRANCE" –source NEWYORK –target DALLAS`**

   Compares the remote SQL Servers ENGLAND and FRANCE accessible with the managed SQL Server NEWYORK against those of the same names for the managed SQL Server DALLAS and displays the differences:

```
<<< Source: NEWYORK master
>>> Target: DALLAS master
---------- ENGLAND ----------
Encrypt Passwords:
< True
> False
Connection Timeout:
< False
> True
Default Remote Login Mapping:
< All-To-One
> None
Default Local Login Name:
< sa
> null
---------- FRANCE ----------
Encrypt Passwords:
< True
Connection Timeout:
< True
Default Remote Login Mapping:
< All-To-One
Default Local Login Name:
< sa
```

2. **`scomprmtserver -names "ENGLAND FRANCE" -source NEWYORK -target DALLAS -remotelogins`**

   Compares the remote SQL Servers ENGLAND and FRANCE accessible to the managed SQL Server NEWYORK with those of the same names for the managed SQL Server DALLAS and displays the differences. In addition to the other remote SQL Server attributes being compared, this command also compares the remote logins associated with each remote SQL Server:

```
<<< Source: NEWYORK
>>> Target: DALLAS
---------- ENGLAND ----------
Encrypt Passwords:
< True
> False
Connection Timeout:
< False
> True
Default Remote Login Mapping:
< All-To-One
> None
Default Local Login Name:
< sa
> null
Remote Logins:
>     Remote Login: arthur
    Local Login Name: guest
    Password Trusted: True
<     Remote Login Name: arthur
    Local Login Name: guest
    Password Trusted: False
<     Remote Login Name: bob
    Local Login Name: guest
    Password Trusted: False
<     Remote Login Name: john
    Local Login Name: pub
    Password Trusted: False
---------- FRANCE ----------
Encrypt Passwords:
< True
Connection Timeout:
< True
Default Remote Login Mapping:
< All-To-One
Default Local Login Name:
< sa
Remote Logins:
<     Remote Login Name: pierre
    Local Login Name: guest
    Password Trusted: False
<     Remote Login Name: jean
    Local Login Name: pub
    Password Trusted: False
```

Comments

- Before using **scomprmtserver**, you need the following information:
  - The names of the remote SQL Servers to compare
  - The names of managed SQL Servers on which to search for the remote SQL Servers
- The **–source** and **–target** SQL Server names must be different. If you specify the same server name for both options, **scomprmtserver** displays an error message.

  Because both **–source** and **–target** options default to the current managed SQL Server collection (see “Command Context” on page 1-15), you must supply at least one of the two SQL Server names. If you omit both options, **scomprmtserver** displays an error message.

Permissions

To use **scomprmtserver**, you must have the following roles:

TME	ESSM	SQL Server
any	any	none

See Also

scopyrmtserver, scrtrmtserver, sdelrmtserver, sgetrmtserver, ssetrmtserver

scomprule

Function

Compares rules from different databases.

Syntax

```
scomprule [-names rule_names] [-bindings]
   [-source db_path] [-target db_path]
   [-wildcard] [-version] [-help]
```

Parameters

–names *rule_names* – specifies the names of the rules to compare. The names of the rules can be prefixed with the name of the owner, as follows: *owner.rule_name*. If you do not specify **–names**, all the rules defined in the database are compared. The name and value are compared.

–bindings – causes the rule's bindings, the list of bound table columns and bound data types, to be compared.

–source *db_path* – specifies the database from which to compare the rules, where *db_path* can be any valid managed database name. The default source is the current managed database. If you do not specify a source database, you must specify a target database using **–target**.

–target *db_path* – specifies the database in which to compare the rules from the source database, where *db_path* can be any valid managed database name. The default target is the current managed database. If you do not specify a target database, you must specify a source database using **–source**.

–wildcard – enables wildcard pattern matching on the rule names.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scomprule -name limit -source ZURICH/pubs2
    -target ZURICH/pubs2_test
```

This command compares the rule *limit* on the database *pubs2* in the managed SQL Server ZURICH with the same rule on the database *pubs2_test* in the managed SQL Server ZURICH.

```
---------- limit dbo ----------
Rule SQL:
Rule text:
< create rule limit as @cash_withdraw < 25100
> create rule limit as @cash_withdraw < 50000
```

Permissions

To use **scomprule**, you must have the following roles:

TME	ESSM	SQL Server
any	any	any

See Also

sgetrule, scrtrule, sdelrule, ssetrule, scopyrule

scompseg

Function

Compares segments between databases and displays the differences. The **scompseg** command compares the names of the devices that the segment spans and, optionally, their thresholds.

Syntax

```
scompseg [–names segment_names [–wildcard]]
   [–thresholds] {[–source db_path] [–target db_path]}
   [–version] [–help]
```

Parameters

–names *segment_names* – specifies which segments to compare, where *segment_names* lists one or more segment names. **scompseg** compares each named segment on the source database with the corresponding segment name on the target database. Without this option, **scompseg** compares all segments occurring on both databases. If the list of segment names contains spaces, you must enclose the list in double quotation marks (see “Option Lists” on page 1-3).

–wildcard – enables wildcard matching on *segment_names* (see “Wildcards” on page 1-4).

–thresholds – compares the thresholds on the segments specified in *segment_names*. Compared attributes include the threshold page limit, threshold procedure name, whether the threshold procedure exists, and last chance threshold status.

–source *db_path* – specifies the database with which to compare *segment_names* on the target database, where *db_path* can be any valid managed database name (see “Database Names” on page 1-16). The default is the current database collection (see “Command Context” on page 1-15). If you do not specify a source database, you must specify a target database using **–target** (see “Comments”). Output lines beginning with “<” refer to the source database.

–target *db_path* – specifies the database with which to compare *segment_names* on the source database, where *db_path* can be any valid managed database name (see “Database Names” on page 1-16). The default is the current database collection (see

"Command Context" on page 1-15). If you do not specify a target database, you must specify a source database using **–source** (see "Comments"). Output lines beginning with ">" refer to the target database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scompseg –names "seg1 seg2" –source pubs2 –target
pubs –threshholds
```

Compares the segments *seg1* and *seg2* in the *pubs2* database against segments of the same names in the *pubs* database, and displays the differences:

```
<<<source: pubs2
>>>target: pubs
----------seg1----------
Devices:
< dev1
> dev2, dev3

----------seg2----------
Devices:
< log_dev
Thresholds: (Pages Left, Proc Name, Proc Exists,
Last Chance)
< 1024, sp_thresholdaction, TRUE, TRUE
> 1024, sp_thresholdaction, FALSE, TRUE
< 2048, seg2_logerror, TRUE, FALSE
```

Comments

- Before using **scompseg**, you need the following information:
  - The particular segment names to be compared
  - The names of the source and target databases on which the compared segments reside
- The **–source** and **–target** database names must be different. If you specify the same database name for both options, **scompseg** displays an error message.

  Because both **–source** and **–target** options default to the current database collection (see "Command Context" on page 1-15), you must supply at least one of the two database names. If you omit both options, **scompseg** displays an error message.
- **scompseg** does not compare schema definitions and data on the segment.

Permissions

To use **scompseg**, you must have the following roles:

TME	ESSM	SQL Server
any	any	**sso_role** or valid database user. You must have **sa_role** to use the **–thresholds** option.

See Also

scompdb, scompdev, scopyseg, scrtseg, sdelseg, sgetseg, ssetseg

scompserver

Function

Compares the configuration variables between managed SQL Servers and reports the differences. See the *Enterprise SQL Server User's Guide* for a summary of SQL Server configuration variables.

Syntax

```
scompserver
   {[–source server_name] [–target server_name]}
   [–version] [–help]
```

Parameters

–source *server_name* – specifies the managed SQL Server with which to compare configuration variables in the target SQL Server, where *server_name* can be any valid managed SQL Server name (see "SQL Server Names" on page 1-15). The default is the current managed SQL Server collection (see "Command Context" on page 1-15). If you do not specify a source SQL Server, you must specify a target SQL Server using **–target** (see "Comments"). Output lines beginning with "<" refer to the source SQL Server.

–target *server_name* – specifies the managed SQL Server with which to compare configuration variables in the source SQL Server, where *server_name* can be any valid managed SQL Server name (see "SQL Server Names" on page 1-15). The default is the current managed SQL Server collection (see "Command Context" on page 1-15). If you do not specify a target SQL Server, you must specify a source SQL Server using **–source** (see "Comments"). Output lines beginning with ">" refer to the target SQL Server.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scompserver –s DEV –t PROD
```

Compares the managed SQL Server named PROD with the managed SQL Server named DEV, and reports the differences:

```
<<<source: DEV
>>>target: PROD
Configuration Options: (Name, Value)
< allow updates, 0
> allow updates, 1
< audit queue size, 100
> audit queue size, 75
< passwordexp, 30
> passwordexp, 0
```

Comments

- The source and target server must be the same release. You cannot compare a release 10.x server to a release 11.x server.
- Before using **scompserver**, you must know the name of the SQL Server installations that you are going to compare.
- The **–source** and **–target** SQL Server names must be different. If you specify the same server name for both options, **scompserver** displays an error message.

  Because both **–source** and **–target** options default to the current managed SQL Server collection (see "Command Context" on page 1-15), you must supply at least one of the two SQL Server names. If you omit both options, **scompserver** displays an error message.

Permissions

To use **scompserver**, you must have the following roles:

TME	ESSM	SQL Server
any	any	none

See Also

smanageserver, ssetserver, scopyserver, sgetserver, sgetprocess, skillprocess, sstartserver, sstopserver

scomptable

Function

Compares tables in a database with tables of the same name in another database.

Syntax

```
scomptable [-names table_names] [-triggers]
   [-indexes] [-bindings]
   [-source db_path] [-target db_path]
   [-wildcard] [-version] [-help]
```

Parameters

–names *table_names* – specifies the names of the tables to compare. The name of the tables can be prefixed with the name of the owner as follows: *owner.table_name*. If you do not specify **–names**, all tables in the specified database are compared. **scomptable** compares table columns, including all components, and segments.

–triggers – compares all triggers and table schema in the specified tables.

–indexes – compares all indexes and table schema in the specified tables.

–bindings – compares the table references, defaults, rules, and datatypes used by the tables and the information returned by sgettable.

–source *db_path* – specifies the database from which to compare the tables, where *db_path* can be any valid managed database name. The default source is the current managed database. If you do not specify a source database, you must specify a target database using **–target**.

–target *db_path* – specifies the database in which to compare the tables from the source database, where *db_path* can be any valid managed database name. The default target is the current managed database. If you do not specify a target database, you must specify a source database using **–source**.

–wildcard – enables wildcard pattern matching on the table names.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scomptable -names salary_tbl
-source PARIS/devel_hrdb
-target ROME/prod_hrdb
```

This command compares the table *salary_tbl* in the database *devel_hrdb* in the managed SQL Server PARIS with the same table in the database *prod_hrdb* in the managed SQL Server ROME.

```
<<< Source: PARIS/devel_hrdb
>>> Target: ROME/prod_hrdb
--------------------------------
---------- salary_tbl ----------
Segment
> public_seg1
< public_seg2
```

Permissions

To use **scomptable**, you must have the following roles:

TME	ESSM	SQL Server
any	any	any

See Also

sgettable, scrttable, sdeltable, ssettable, scopytable, schecktable

scomptrigger

Function

Compares triggers in a database with triggers of the same name in another database.

Syntax

```
scomptrigger [-names trigger_names]
   [-source db_path] [-target db_path]
   [-wildcard] [-version] [-help]
```

Parameters

–names *trigger_names* – specifies the names of the triggers to compare. The names of the triggers can be prefixed with the name of the owner as follows: owner.trigger_name. If you do not specify **–names**, all the triggers defined in the database are compared. The trigger schema including name, type, and SQL are compared.

–source *db_path* – specifies the database from which to compare the triggers, where *db_path* can be any valid managed database name. The default source is the current managed database. If you do not specify a source database, you must specify a target database using **–target**.

–target *db_path* – specifies the database in which to compare the triggers from the source database, where *db_path* can be any valid managed database name. The default target is the current managed database. If you do not specify a target database, you must specify a source database using **–source**.

–wildcard – enables wildcard pattern matching on the trigger names.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scomptrigger -name dbo.deltitle
-source BOSTON/pubs2
-target PARIS/pubs2_test
```

This command compares the trigger named *deltitle* in the database *pubs2* in the managed SQL Server BOSTON with the same trigger in the database *pubs2_test* in the managed SQL Server BOSTON.

```
---------- deltitle dbo ----------
Trigger Create Date:
< Oct 31 1995  9:46AM
> Oct 23 1995 11:49AM
Trigger SQL:
Trigger text:
<
/*** Create some standard triggers */
create trigger deltitle
on titles
for delete
as
print "You can't delete a title."

>
/*** Create some standard triggers */
create trigger deltitle
on titles
for delete
as
    if (select count(*) from deleted, salesdetail
    where salesdetail.title_id = deleted.title_id)
>0
    begin
        rollback transaction
        print "You can't delete a title with
Trigger text:
>  sales."
    end
```

Permissions

To use **scomptrigger**, you must have the following roles:

TME	ESSM	SQL Server
any	any	any

See Also

sgettrigger, scrttrigger, sdeltrigger, scopytrigger

scompuser

Function

Compares database users between databases and displays the differences. The **scompuser** command compares the following user attributes:

- Login name
- Group name
- Login aliases (optional)

Syntax

```
scompuser [–names user_names [–wildcard]]
   [–aliases] {[–source db_path] [–target db_path]}
   [–version] [–help]
```

Parameters

–names *user_names* – specifies which database users to compare, where *user_names* lists one or more database user_names. **scompuser** compares each named user on the source database with the corresponding name on the target database. Without this option, **scompuser** compares all users occurring on both databases. If the list of user_names contains spaces, you must enclose the list in double quotation marks (see “Option Lists” on page 1-3).

–wildcard – enables wildcard matching on *user_names* (see “Wildcards” on page 1-4).

–aliases – includes user aliases in the user attributes being compared.

–source *db_path* – specifies the database with which to compare *user_names* on the target database, where *db_path* can be any valid managed database name (see “Database Names” on page 1-16). The default is the current database collection (see “Command Context” on page 1-15). If you do not specify a source database, you must specify a target database using **–target** (see “Comments”). Output lines beginning with “<” refer to the source database.

–target *db_path* – specifies the database with which to compare *user_names* on the source database, where *db_path* can be any valid managed database name (see “Database Names” on page 1-16). The default is the current managed database. If you do not specify a target database, you must specify a source database using **–source** (see “Comments”). Output lines beginning with “>” refer to the target database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scompuser –source SNIBOR/master –target
CHARPOY/master
```

Compares all users in the *master* database on the managed SQL Server SNIBOR with those in the *master* database on the managed SQL Server CHARPOY and displays the differences:

```
<<<source: SNIBOR/master
>>>target: CHARPOY/master
----------harryc----------
Login Name:
< harryc
> harry
----------bmiley----------
Group Name:
< public
> hr
```

Comments

- Before using **scompuser**, you need the following information:
  - The usernames to compare
  - The names of the databases on which the users reside
- The **–source** and **–target** database names must be different. If you specify the same server name for both options, **scompuser** displays an error message.

  Because both **–source** and **–target** options default to the current database collection (see “Command Context” on page 1-15), you must supply at least one of the two database names. If you omit both options, **scompuser** displays an error message.

Permissions

To use **scompuser**, you must have the following roles:

TME	ESSM	SQL Server
any	any	**sso_role**, or **sa_role**, or valid database user

See Also

scopyuser, scrtuser, sdeluser, sgetuser, ssetgroup, ssetuser

scompview

Function

Compares views in a database with views in another database.

Syntax

```
scompview [-names view_names] [-source db_path]
   [-target db_path] [-wildcard] [-version] [-help]
```

Parameters

–names *view_names* – specifies the names of the views to compare. The names of the views can be prefixed with the name of the owner, as follows: *owner.view_name*. If you do not specify **–names**, all the views defined in the database are compared. The command compares SQL statements for selecting views.

–source *db_path* – specifies the database from which to compare the views, where *db_path* can be any valid managed database name. The default source is the current managed database. If you do not specify a source database, you must specify a target database using **–target**.

–target *db_path* – specifies the database in which to compare the views from the source database, where *db_path* can be any valid managed database name. The default target is the current managed database. If you do not specify a target database, you must specify a source database using **–source**.

–wildcard – enables wildcard pattern matching on the view names.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scompview -name dbo.titleview
-source PARIS/pubs2 -targ BOSTON/pubs2
```

This command compares the view *titleview*, owned by *dbo*, in the database *pubs2* in the managed SQL Server PARIS with the same view in the database *pubs2* in the managed SQL Server BOSTON.

```
---------- titleview ----------
Summary Text:
View text:
< = titleauthor.au_id
        and titles.title_id = titleauthor.title_id
>
create view titleview
as
select title, au_ord, au_lname,
price, total_sales, pub_id
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
Owner Name:
< dbo
View text:
<
/*
** Create a few views now.  Start with a view
combining the authors,
** titles, and titleauthors tables
*/
create view titleview
as
        select title, au_ord, au_lname,
        price, total_sales, pub_id
        from authors, titles, titleauthor
        where authors.au_id
```

Permissions

To use **scompview**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	any

See Also

sgetview, scrtview, sdelview, scopyview

scopycmdperm

Function

Copies command permissions from one database to another. The **scopycmdperm** command copies permission information for the following commands:

- **create database**
- **create default**
- **create procedure**
- **create rule**
- **create table**
- **create view**
- **dump database**
- **dump transaction**

Syntax

```
scopycmdperm
   [-names users_or_groups_or_role_names [-wildcard]]
   {[-source db_path] [-target db_path]}
   [-version] [-help]
```

Parameters

–names *users*_or_*groups*_or_*role_names* – specifies the user, group, or role for which to copy permission information.

–wildcard – enables use of wildcard characters in the user, group, or role names (see "Wildcards" on page 1-4).

–source *db_path* – specifies the database to copy, where *db_path* can be any valid managed database name (see "Database Names" on page 1-16). The default is the current database collection (see "Command Context" on page 1-15).

–target *db_path* – specifies the new database name, where *db_path* can be any valid managed database name (see "Database Names" on page 1-16). You must specify a target database. The target database must not already exist before the copy operation (see "Comments").

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scopycmdperm -names bennyg -source AGRA/pubs2
-target KOKO/pubs2
```

Copies the command permissions of user *bennyg* from the *pubs2* database on the managed Server AGRA to the *pubs2* database on the managed Server KOKO.

Comments

- Before using **scopycmdperm**, you need the following information:
  - The names of the databases being compared
  - The names of any user or group that you want to compare
- If an error occurs, **scopycmdperm** rolls back the entire copy operation, restoring all affected objects to their original state.
- **scopycmdperm** does not copy implicit permissions. It only copies command permissions that were set explicitly with the **ssetcmdperm**.

Permissions

To use **scopycmdperm**, you must have the following roles:

TME	ESSM	SQL Server
any	**security**	**sso_role**

See Also

scopyobjperm, sgetcmdperm

scopydatatype

Function

Copies user datatypes from one database to another.

Syntax

```
scopydatatype [-names datatype_names] [-bindings]
   [-source db_path] [-target db_path]
   [-wildcard] [-version] [-help]
```

Parameters

–names *datatype_names* – specifies the names of the datatypes to copy. The names of the datatypes can be prefixed with the name of the owner as follows: owner.type_name. If you do not specify **–names**, all the datatypes defined in the database are copied. The following information will be copied:

- physical_type
- length
- precision
- scale
- identity / null / nonull setting

–bindings – causes the datatype's bindings, the bound rule name, and bound default name to be copied. If the rule or default does not exist within the target database, the copy fails.

–source *db_path* – specifies the database from which to copy the datatypes, where *db_path* can be any valid managed database name. The default source is the current managed database. If you do not specify a source database, you must specify a target database using **–target**.

–target *db_path* – specifies the database in which to copy the datatypes from the source database, where *db_path* can be any valid managed database name. The default target is the current managed database. If you do not specify a target database, you must specify a source database using **-source**.

–wildcard – enables wildcard pattern matching on the datatype names.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scopydatatype -names ssn -source BONN/devel_hrdb
-target ROME/prod_hrdb
```

This command copies the datatype *ssn* from the database *devel_hrdb*, in the managed SQL Server BONN, to the database *prod_hrdb*, in the managed SQL Server ROME.

Permissions

To use **scopydatatype**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	any

See Also

sgetdatatype, scrtdatatype, sdeldatatype, ssetdatatype, scompdatatype

scopydb

Function

Copies one database to another database, on the same SQL Server or between different managed servers. The **scopydb** command copies:

- The Database Owner
- Devices
- Configuration options

Syntax

```
scopydb [–source db_path] –target db_path
   [–datadevices "{DEFAULT | device_name} device_size
       [, device size]...]"]
   [–logdevices "{DEFAULT | device_name} device_size
       [, device size]...]"]
   [–override] [–forload] [–version] [–help]
```

Parameters

–source *db_path* – specifies the database to copy, where *db_path* can be any valid managed database name (see “Database Names” on page 1-16). The default is the current database collection (see “Command Context” on page 1-15).

–target *db_path* – specifies the new database name, where *db_path* can be any valid managed database name (see “Database Names” on page 1-16). You must specify a target database. The target database must not already exist before the copy operation (see “Comments”).

–datadevices – specifies the database devices (**default** or *device*) and the corresponding space (*size*) on each device to allocate to the target database for storing data. If you omit this option, **scopydb** uses the same devices and space allocation as the source database. You must enclose the list of *device size* pairs in double quotation marks (see “Option Lists” on page 1-3).

DEFAULT – indicates that **scopydb** can put the new database on any of the target SQL Server’s default database devices (listed in *sysdevices.status*).

device_name – specifies the logical name of a device on which the new database may store information. A database can occupy different amounts of space on each of several database devices, thus you can specify multiple *device size* pairs in the argument list.

device_size – specifies the amount of space (in MB) to allocate on the specified device (**default** or *device*).

–logdevices – specifies the database devices (**default** or *device*) and the corresponding space (*size*) on each device to allocate to the target database for storing the transaction log. You must enclose the list of *device size* pairs in double quotation marks. If you omit this option, **scopydb** uses the same devices and space allocation as the source database for storing the transaction log.

–override – allows you to specify the same device name for both the data and the transaction log. This option allows SQL Server on a machine with limited space to still maintain the transaction log on separate device fragments from the database's data. Without this option, the **scrtdb** command fails if you attempt to mix the data and transaction log on the same device.

–forload – creates a target database for use only in loading a database dump. Use this option for recovery from media failure or for moving a database from one machine to another.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`scopydb –source db1 –target db2`**

   Copies database *db1* to new database *db2* on the current SQL Server.

2. **`scopydb –s BOSTON/model –t EMERY/model`**

   Copies the *model* database from the BOSTON SQL Server to the EMERY SQL Server.

Comments

- Before using **scopydb**, you must know the name of the target database to create.
- If you specify a device (*device* or DEFAULT), you must also include a corresponding *size* value.
- If the devices are not specified, **scopydb** uses the default devices on the target.
- If an error occurs, **scopydb** rolls back the entire copy operation, restoring all affected objects to their original state.
- The copy operation fails and **scopydb** displays an error message if:
  - The specified devices do not exist on the target SQL Server
  - The owner of the source database does not exist as a login in the target database's SQL Server
  - The target database already exists when you issue the **scopydb** command
- If SQL Server cannot give you as much space as you want where you have requested it, it comes as close as possible on a per-device basis, and prints a message telling how much space was allocated where.
- **scopydb** does not copy database objects (such as tables, views, users, stored procedures), only databases.
- To modify remote SQL Server attributes, use **scopydb**.

Permissions

To use **scopydb**, you must have the following roles:

TME	ESSM	SQL Server
any	**space**	**sa_role**

See Also

scheckdb, scompdb, scopydev, scopygroup, scopyuser, scrtdb, sdeldb, sgetdb, ssetdb

scopydefault

Function

Copies default column values from one database to other.

Syntax

```
scopydefault [-names default_names] [-bindings]
   [-source db_path ] [-target db_path ] [-wildcard]
   [-version] [-help]
```

Parameters

–names *default_names* – specifies the names of the defaults to copy. The names of the defaults can be prefixed with the name of the owner as follows: owner.default_name. If you do not specify **–names**, all the defaults defined in the database will be copied. The following information will be copied:

- name
- owner
- value

–bindings – specifies that any bindings to user datatypes and columns be copied. Replaces any existing column and datatype bound defaults. If any attempted binding fails, the whole command fails.

–source *db_path* – specifies the database from which to copy the defaults, where *db_path* can be any valid managed database name. The default source is the current managed database. If you do not specify a source database, you must specify a target database using **–target**.

–target *db_path* – specifies the database in which to copy the defaults from the source database, where *db_path* can be any valid managed database name. The default target is the current managed database. If you do not specify a target database, you must specify a source database using **–source**.

–wildcard – enables wildcard pattern matching on the default names.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scopydefault -names sex_def -source BONN/dev_hrdb
-target ROME/prod_hrdb
```

This command copies the default *sex_def* from the database *dev_hrdb*, in the managed SQL Server BONN, to the database *prod_hrdb*, in the managed SQL Server ROME.

Permissions

To use **scopydefault**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	Database Owner. If you are specifying bindings, you must be the owner of the object being bound to.

See Also

sgetdefault, scrtdefault, sdeldefault, ssetdefault, scompdefault

scopydev

Function

Copies devices from one managed SQL Server to another. The **scopydev** command copies the following device attributes:

- Logical device name
- Physical device name
- Size
- Default pool
- Mirror device name
- Mirror status
- Starting virtual address
- Disk controller
- Virtual device number

Syntax

```
scopydev [–names device_names [–wildcard]]
   {[–source server_name] [–target server_name]}
   [–version] [–help]
```

Parameters

–names *device_names* – specifies which database devices to copy, where *device_names* lists one or more database device names existing in the source SQL Server. **scopydev** copies each named device in the source SQL Server to an identical name in the target SQL Server. Without this option, **scopydev** copies all database devices occurring in the source SQL Server to the target SQL Server. Separate each device name with a space and enclose the list in double quotation marks.

–wildcard – enables wildcard matching on *device_names* (see “Wildcards” on page 1-4).

–source *server_name* – specifies the managed SQL Server in which the existing *device_names* reside, where *server_name* can be any valid managed SQL Server name (see “SQL Server Names” on page 1-15). The default is the current managed SQL Server collection (see “Command Context” on page 1-15). If you do not specify a source SQL Server, you must specify a target SQL Server using **–target** (see “Comments”).

–target *server_name* – specifies the managed SQL Server in which to copy *device_names*, where *server_name* can be any valid managed SQL Server name (see “SQL Server Names” on page 1-15). The default is the current managed SQL Server collection (see “Command Context” on page 1-15). If you do not specify a target SQL Server, you must specify a source SQL Server using **–source** (see “Comments”).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`scopydev –names master –source BOSTON –target EMERY`**

   Copies the existing *master* device in the BOSTON managed SQL Server to the EMERY managed SQL Server. The device name remains the same.

2. **`scopydev –na "dev1 dev2 logdev1"`**
   `–t NEsalesRegion/SQLservers/EMERY`

   Copies the *dev1*, *dev2*, and *logdev1* devices in the current managed SQL Server to the *NEsalesRegion/SQLservers/EMERY* managed SQL Server.

Comments

- Before using **scopydev**, you need the following information:
  - The source and target managed SQL Server names
  - The names of the devices to copy
- The **–source** and **–target** SQL Server names must be different. If you specify the same SQL Server name for both options, **scopydev** displays an error message.

  Because both **–source** and **–target** options default to the current managed SQL Server collection (see “Command Context” on page 1-15), you must supply at least one of the two SQL Server names. If you omit both options, **scopydev** displays an error message.
- If an error occurs, **scopydev** rolls back the entire copy operation, restoring all affected objects to their original state.

- If a specified device name already exists in the target SQL Server, the **scopydev** command fails and displays an error message.

Permissions

To use **scopydev**, you must have the following roles:

TME	ESSM	SQL Server
any	**space**	**sa_role**

See Also

scrtdev, scompdev, sdeldev, sgetdev, ssetdev

scopydumpdev

Function

Copies dump devices from one managed SQL Server to another. The **scopydumpdev** command copies the following device attributes:

- Logical device name
- Physical device name
- Tape / disk kind
- Size

Syntax

```
scopydumpdev [-names dump_device_names [-wildcard]]
   {[-source server_name] [-target server_name]}
   [-version] [-help]
```

Parameters

–names *dump_device_names* – specifies which dump devices to copy, where *dump_device_names* lists one or more dump device names existing in the source SQL Server. **scopydumpdev** copies each named device in the source SQL Server to an identical name in the target SQL Server. Without this option, **scopydumpdev** copies all dump devices in the source SQL Server to the target SQL Server. If the list of device names contains spaces, you must enclose the list in double quotation marks.

–wildcard – enables wildcard matching on *dump_device_names* (see “Wildcards” on page 1-4).

–source *server_name* – specifies the managed SQL Server in which the existing *device_names* reside, where *server_name* can be any valid managed SQL Server name (see “SQL Server Names” on page 1-15). The default is the current managed SQL Server collection (see “Command Context” on page 1-15). If you do not specify a source SQL Server, you must specify a target SQL Server using **–target** (see “Comments”).

–target *server_name* – specifies the managed SQL Server in which to copy *device_names*, where *server_name* can be any valid managed SQL Server name (see “SQL Server Names” on page 1-15). The default is the current managed SQL Server collection (see “Command Context” on page 1-15). If you do not specify a target

SQL Server, you must specify a source SQL Server using **–source** (see “Comments”).

–version – prints the release and copyright of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scopydumpdev -target KOKO
```

Copies the dump devices from the current managed SQL Server to the managed SQL Server KOKO.

Comments

- Before using **scopydumpdev**, you must have the following information:
  - The names of the managed Servers that control the devices.
  - The names of any dump devices that you want to copy.
- The **–source** and **–target** SQL Server names must be different. If you specify the same SQL Server name for both options, **scopydumpdev** displays an error message.

  Because both options default to the current managed SQL Server collection (see “Command Context” on page 1-15), you must supply at least one of the two SQL Server names.
- If an error occurs, **scopydumpdev** rolls back the entire copy operation, deleting any devices that it previously successfully added. This will normally restore the original state, unless something unusual happens, such as concurrent activity, or a machine or network crash during the operation (see “Transactions”).
- If a specified device name already exists in the target SQL Server, the **scopydumpdev** command fails and displays an error message.

Permissions

To use **scopydumpdev**, you must have the following roles:

TME	ESSM	SQL Server
any	**space**	**sa_role**

See Also

scrtdumpdev, sdeldumpdev, scompdumpdev, sgetdumpdev

scopygroup

Function

Copies database user groups from one database to another. The **scopygroup** command copies only the specified groups, not the users in those groups (see **scopyuser**).

Syntax

```
scopygroup [–names group_names]
   {[–source db_path] [–target db_path]}
   [-permissions] [–wildcard] [–version] [–help]
```

Parameters

–names *group_names* – specifies which database groups to copy, where *group_names* lists one or more group names existing in the source database. **scopygroup** copies each named group in the source database to an identical name in the target database. Without this option, **scopygroup** copies all groups occurring on the source database to the target database. If the list of group names contains spaces, you must enclose the list in double quotation marks (see “Option Lists” on page 1-3).

–wildcard – enables wildcard matching on *group_names* (see “Wildcards” on page 1-4).

–source *db_path* – specifies the database from which to copy *group_names*, where *db_path* can be any valid managed database name (see “Database Names” on page 1-16). The default is the current database collection (see “Command Context” on page 1-15). If you do not specify a source database, you must specify a target database using **–target** (see “Comments”).

–target *db_path* – specifies the database on which to copy *group_names* from the source database, where *db_path* can be any valid managed database name (see “Database Names” on page 1-16). The default is the current database collection (see “Command Context” on page 1-15). If you do not specify a target database, you must specify a source database using **–source** (see “Comments”).

–permissions – copies the command permissions associated with the groups you are copying.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`scopygroup –names clerk –source accounts –target inventory`**

   Copies the group *clerk* from the *accounts* database to the *inventory* database in the current managed SQL Server.

2. **`scopygroup –n "eng qa" –s rel9 –t rel10`**

   Copies the groups *eng* and *qa* from the *rel9* database to the *rel10* database on the current managed SQL Server.

Comments

- Before using **scopygroup**, you need the following information:
  - The names of source and target databases
  - The names of the groups to copy
- The **–source** and **–target** database names must be different. If you specify the same database name for both options, **scopygroup** displays an error message.

  Because both **–source** and **–target** options default to the current database collection (see “Command Context” on page 1-15), you must supply at least one of the two database names. If you omit both options, **scopygroup** displays an error message.
- If an error occurs, **scopygroup** rolls back the entire copy operation, restoring all affected objects to their original state.
- If a specified group name already exists on the target database, the **scopygroup** command fails and displays an error message.

Permissions

To use **scopygroup**, you must have the following roles:

TME	ESSM	SQL Server
any	**security**	**sa_role** or Database Owner

See Also

scompgroup, scopyuser, scrtgroup, sdelgroup, sgetgroup, ssetgroup

scopyindex

Function

Copies index definitions to a table of the same name in another database.

Syntax

```
scopyindex [-names index_names] [-source db_path]
   [-target db_path] [-wildcard] [-version] [-help]
```

Parameters

–names *index_names* – specifies the names of the indexes to copy in the format owner.table.indexname. If you do not specify **–names**, all indexes defined in the default or specified database are copied.

Any of the three parts of *index_names* can be replaced with a wildcard. The owner and table name are optional.

The following information is copied:

- name
- owner (the new owner will be the same as the old owner - ownership is always copied exactly)
- type
- duplicate key setting
- duplicate row setting
- segment
- columns

–source *db_path* – specifies the database from which to copy the indexes, where *db_path* can be any valid managed database name. The default source is the current managed database. If you do not specify a source database, you must specify a target database using **–target**.

–target *db_path* – specifies the database in which to copy the indexes from the source database, where *db_path* can be any valid managed database name. The default target is the current managed database. If you do not specify a target database, you must specify a source database using **–source**.

–wildcard – enables wildcard pattern matching on the index names.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scopyindex -names salary -source BONN/devel_hrdb
-target ROME/prod_hrdb
```

This command copies the index *salary* from the table *salary_table* in the source database *devel_hrdb* in the managed SQL Server BONN to the target database *prod_hrdb* in the managed SQL Server ROME.

Comment

You can omit the first two parts (owner and table name) of the index_names argument. The name "a.b" implies that the owner was omitted. The name "a" implies that both the owner and table were omitted. If you just specify the index name, all indexes named with that name are retrieved, regardless of their owner or the table they are defined on.

Permissions

To use **scopyindex**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	owner of target table

See Also

scheckindex, sgetindex, scrtindex, sdelindex, ssetindex, scompindex

scopylogin

Function

Copies logins from one managed SQL Server to another. The **scopylogin** command copies the following login attributes:

- Full name
- Database
- Language
- Lock status
- Roles

Syntax

```
scopylogin [–names login_names [–wildcard]]
   {[–source server_name] [–target server_name]}
   [–version] [–help]
```

Parameters

–names *login_names* – specifies which login names to copy, where *login_names* lists one or more login names existing on the source SQL Server. **scopylogin** copies each named login on the source SQL Server to an identical name on the target SQL Server. Without this option, **scopylogin** copies all logins occurring on the source SQL Server. Separate the login names with a space and enclose the list in double quotation marks (see "Option Lists" on page 1-3).

–wildcard – enables wildcard matching on *login_names* (see "Wildcards" on page 1-4).

–source *server_name* – specifies the managed SQL Server from which to copy *login_names*, where *server_name* can be any valid managed SQL Server name (see "SQL Server Names" on page 1-15). The default is the current managed SQL Server collection (see "Command Context" on page 1-15). If you do not specify a source SQL Server, you must specify a target SQL Server using **–target** (see "Comments").

–target *server_name* – specifies the managed SQL Server on which to copy *login_names* from the source SQL Server, where *server_name* can be any valid managed SQL Server name (see "SQL Server Names" on page 1-15). The default is the current managed SQL Server collection (see "Command Context" on page 1-15). If you do not specify a target SQL Server, you must specify a source SQL Server using **–source** (see “Comments”).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`scopylogin –names david –target CONCORD`**

   Copies the user login *david* from the current managed SQL Server to the managed SQL Server CONCORD.

2. **`scopylogin –s BOSTON –T BURL`**

   Copies all user logins from the BOSTON SQL Server to the BURL SQL Server, using the password specified in the policy default.

Comments

- Before using **scopylogin**, you need the following information:
  - The login names to copy
  - The names of the source and target servers
- The **–source** and **–target** SQL Server names must be different. If you specify the same SQL Server name for both options, **scopylogin** displays an error message.

  Because both **–source** and **–target** options default to the current managed SQL Server collection (see “Command Context” on page 1-15), you must supply at least one of the two SQL Server names. If you omit both options, **scopylogin** displays an error message.
- If an error occurs, **scopylogin** rolls back the entire copy operation, restoring all affected objects to their original state.
- If any of the specified login names already exist on the target SQL Server, the **scopylogin** command fails.
- To modify passwords and other login attributes, use the **ssetlogin** command.
- To modify login auditing information, use the **ssetloginaudit** command.
- To modify remote SQL Server attributes, use the **ssetrmtserver** command.

- To copy database users associated with a SQL Server login, use the **scopyuser** command. Copy the login before copying the user.
- You cannot copy encrypted SQL Server login passwords between a SunOS Release 4.x system and a system using a different operating system.

Permissions

To use **scopylogin**, you must have the following roles:

TME	ESSM	SQL Server
any	**security**	**sso_role** and **sa_role**

See Also

scomplogin, scopyuser, scrtlogin, sdellogin, sgetlogin, ssetlogin, ssetloginaudit, ssetrmtserver

scopyobjperm

Function

Copies object permissions that have been explicitly set with the **ssetobjperm** command from one database to another. The **scopyobjperm** command copies permission information on the following objects:

- tables
- views
- procedures

Syntax

```
scopyobjperm [–names object_names [–wildcard]]
   {[–source db_path] [–target db_path]}
   [–version] [–help]
```

Parameters

–names *object_names* – specifies the objects for which to copy permission information.

–wildcard – enables wildcard matching on *object_names* (see “Wildcards” on page 1-4).

–source *db_path* – specifies the database to copy, where *db_path* can be any valid managed database name (see “Database Names” on page 1-16). The default is the current database collection (see “Command Context” on page 1-15).

–target *db_path* – specifies the new database name, where *db_path* can be any valid managed database name (see “Database Names” on page 1-16). You must specify a target database. The target database must not already exist before the copy operation (see “Comments”).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scopyobjperm -names "records performances" -source
AGRA/pubs2 -target KOKO/jazz
```

Copies the object permissions on the tables *records* and *performances* from the *pubs2* database on the managed Server AGRA to the *jazz* database on the managed Server KOKO.

Comments

- Before using **scopyobjperm**, you need the following information:
  - The source or target database
  - The names of specific objects whose object permissions you want to copy
- If an error occurs, **scopyobjperm** rolls back the entire copy operation, restoring all affected objects to their original state.

Permissions

To use **scopyobjperm**, you must have the following roles:

TME	ESSM	SQL Server
any	**security**	**sso_role**

See Also

scheckdb, scompdb, scopydev, scopygroup, scopyuser, scrtdb, sdeldb, sgetdb, ssetdb

scopyproc

Function

Copies stored procedures from one database to another.

Syntax

```
scopyproc [-names procedure_names] [-permissions]
   [-source db_path] [-target db_path]
   [-wildcard] [-version] [-help]
```

Parameters

–names *procedure_names* – specifies the names of the stored procedures to copy. The names of the stored procedures can be prefixed with the name of the owner as follows: owner.procedure_name. If you do not specify **–names**, all the stored procedures defined in the database are copied. The following information is copied: stored procedure schema including name, owner, parameters, recompile status, and SQL.

–permissions – copies the object permissions associated with the stored procedures you are copying.

–source *db_path* – specifies the database from which to copy the stored procedures, where *db_path* can be any valid managed database name. The default source is the current managed database. If you do not specify a source database, you must specify a target database using **–target**.

–target *db_path* – specifies the database in which to copy the stored procedures from the source database, where *db_path* can be any valid managed database name. The default target is the current managed database. If you do not specify a target database, you must specify a source database using **–source**.

–wildcard – enables wildcard pattern matching on the stored procedure names.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

➤ ***Note***

Some procedures may depend on other procedures. **scopyproc** copies procedures in the names list in the proper order for their dependencies, but the command does not know if a prerequisite procedure is missing from the list. You must be sure that the list includes all necessary procedures. If you copy a procedure without one of its prerequisite procedures, it will not work in the target database.

Example

```
scopyproc -names locked_logins
-source BOSTON/salesQ195 -target BOSTON/salesQ295
```

This command copies the procedure *locked_logins* from the database *salesQ195,* in the managed SQL Server BOSTON to the database *salesQ295,* in the managed SQL Server BOSTON.

Permissions

To use **scopyproc**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	Database Owner of the target database

See Also

sgetproc, scrtproc, sdelproc, scompproc

scopyrmtserver

Function

Copies remote SQL Server entries and options from one managed SQL Server to another. The **scopyrmtserver** command copies the following remote SQL Server attributes:

- Timeout configuration
- Password encryption configuration
- Local login mapping on the remote SQL Server

Optionally, **scopyrmtserver** can copy remote logins.

Syntax

```
scopyrmtserver
   [–names remote_server_names [–wildcard]]
   [–remotelogins]
   {[–source server_name] [–target server_name]}
   [–version] [–help]
```

Parameters

–names *remote_server_names* – specifies which remote SQL Server entries to copy, where *remote_server_names* lists one or more remote SQL Server names. **scopyrmtserver** copies each named remote SQL Server on the source SQL Server to an identical name on the target SQL Server. Without this option, **scopyrmtserver** copies all remote servers occurring on the source SQL Server. If the list of remote SQL Server names contains spaces, you must enclose the list in double quotation marks (see "Option Lists" on page 1-3).

–wildcard – enables wildcard matching on *remote_server_names* (see "Wildcards" on page 1-4).

–remotelogins – copies the remote logins associated with each remote SQL Server, in addition to the other remote SQL Server names being copied.

–source *server_name* – specifies the managed SQL Server from which to copy *remote_server_names*, where *server_name* can be any valid managed SQL Server name (see "SQL Server Names" on page 1-15). The default is the current managed SQL Server collection (see "Command Context" on page 1-15). If you do not specify a source SQL Server, you must specify a target SQL Server using **–target** (see “Comments”).

–target *server_name* – specifies the managed SQL Server on which to copy *remote_server_names* from the source SQL Server, where *server_name* can be any valid managed SQL Server name (see “SQL Server Names” on page 1-15). The default is the current managed SQL Server collection (see “Command Context” on page 1-15). If you do not specify a target SQL Server, you must specify a source SQL Server using **–source** (see “Comments”).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`scopyrmtserver –names FRANCE –target CHICAGO`**

   Copies the remote SQL Server entry FRANCE from the current managed SQL Server to the managed SQL Server CHICAGO.

2. **`scopyrmtserver –s BOSTON –t CENTRAL`**

   Copies all remote SQL Server entries in the managed SQL Server BOSTON to the managed SQL Server CENTRAL.

3. **`scopyrmtserver –names %SALES –wildcard –t CENTRAL`**

   Copies all remote SQL Server entry names ending with “SALES” (for example, *NE_SALES*, *SW_SALES*, *EURO_SALES*, and so on) from the current managed SQL Server to the managed SQL Server CENTRAL.

Comments

- Before using **scopyrmtserver**, you need the following information:

  -The remote SQL Server names to copy

  -The names of the source and target servers

- The **–source** and **–target** SQL Server names must be different. If you specify the same SQL Server name for both options, **scopyrmtserver** displays an error message.

  Because both **–source** and **–target** options default to the current managed SQL Server collection (see “Command Context” on page 1-15), you must supply at least one of the two SQL Server names. If you omit both options, **scopyrmtserver** displays an error message.

- If an error occurs, **scopyrmtserver** rolls back the entire copy operation, restoring all affected objects to their original state.
- If a remote SQL Server name already exists on the target SQL Server, the **scopyrmtserver** command fails and displays an error message.
- The **scopyrmtserver** command does not copy actual servers, only the remote SQL Server entries (appearing in *sysservers*) from the source SQL Server.
- To modify remote SQL Server attributes, use the **ssetrmtserver** command.

Permissions

To use **scopyrmtserver**, you must have the following roles:

TME	ESSM	SQL Server
any	**security**	**sso_role** and **sa_role**

See Also

scomprmtserver, scrtrmtserver, sdelrmtserver, sgetrmtserver, ssetrmtserver

scopyrule

Function

Copies rules from one database to another.

Syntax

```
scopyrule [-names rule_names] [-bindings]
   [-source db_path] [-target db_path]
   [-wildcard] [-version] [-help]
```

Parameters

–names *rule_names* – specifies the names of the rules to copy. The names of the rules can be prefixed with the name of the owner as follows: owner.rule_name. If you do not specify **–names**, all the rules defined in the database will be copied. The following information will be copied:

- name
- owner
- value

–bindings – copies any rule bindings to default types and table columns. If any attempted binding fails, the whole command will fail.

–source *db_path* – specifies the database from which to copy the rules, where *db_path* can be any valid managed database name. The default source is the current managed database. If you do not specify a source database, you must specify a target database using **–target**.

–target *db_path* – specifies the database in which to copy the rules from the source database, where *db_path* can be any valid managed database name. The default target is the current managed database. If you do not specify a target database, you must specify a source database using **–source**.

–wildcard – enables wildcard pattern matching on the rule names.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scopyrule -names limit -source PARIS/transactions
-target ROME/transactions
```

This command copies the rule *limit* from the database *transactions*, in the managed SQL Server PARIS to the database *transactions*, in the managed SQL Server ROME.

Permissions

To use **scopyrule**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	Database Owner of the target database. If you are specifying bindings, you must be the owner of the object being bound to.

See Also

sgetrule, scrtrule, sdelrule, ssetrule, scomprule

scopyseg

Function

Copies segments from one database to another. The **scopyseg** command copies the device names on which the segment is stored, and optionally the thresholds.

Syntax

```
scopyseg [–names segment_names [–wildcard]]
   [–thresholds] {[–source db_path] [–target db_path]}
   [–version] [–help]
```

Parameters

–names *segment_names* – specifies which segments to copy, where *segment_names* lists one or more segment names in the source database. **scopyseg** copies each named segment in the source database to an identical segment name in the target database. Without this option, **scopyseg** copies all segments occurring on the source database. If the list of segment names contains spaces, you must enclose the list in double quotation marks (see "Option Lists" on page 1-3).

–wildcard – enables wildcard matching on *segment_names* (see "Wildcards" on page 1-4).

–thresholds – copies the thresholds on the specified segments, including the free page limit, the procedure name to be executed, and last chance status.

–source *db_path* – specifies the database from which to copy *segment_names*, where *db_path* can be any valid managed database name (see "Database Names" on page 1-16). The default is the current database collection (see "Command Context" on page 1-15). If you do not specify a source database, you must specify a target database using **–target** (see "Comments").

–target *db_path* – specifies the database in which to copy *segment_names* from the source database, where *db_path* can be any valid managed database name (see "Database Names" on page 1-16). The default is the current database collection (see "Command Context" on page 1-15). If you do not specify a target database, you must specify a source database using **–source** (see "Comments").

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`scopyseg –name logseg –target pubs2 –thresholds`**

   Copies the segment *logseg* and its thresholds from the current database to the *pubs2* database in the current managed SQL Server.

2. **`scopyseg –source BOSTON/pubs2 –target CENTRAL/newpubs`**

   Copies all segments from the *pubs2* database on the managed SQL Server BOSTON to the *newpubs* database on the managed SQL Server CENTRAL.

3. **`scopyseg –n "seg1 seg2" –s master –target pubs`**

   Copies the segments *seg1* and *seg2* from the *master* database to the *pubs* database in the current managed SQL Server.

Comments

- Before using **scopyseg**, you need the following information:
  - The segment names to copy
  - The names of the source and target databases
- The **–source** and **–target** database names must be different. If you specify the same database name for both options, **scopyseg** displays an error message.

  Because both **–source** and **–target** options default to the current database collection (see “Command Context” on page 1-15), you must supply at least one of the two database names. If you omit both options, **scopyseg** displays an error message.
- If an error occurs, **scopyseg** rolls back the entire copy operation, restoring all affected objects to their original state.
- If a segment name already exists in the target database, the **scopyseg** command fails and displays an error message.
- To modify segment attributes, use the **ssetseg** command.
- The **scopyseg** command does not copy segment schema definitions.

Permissions

To use **scopyseg**, you must have the following roles:

TME	ESSM	SQL Server
any	**space**	**sa_role**

See Also

scompseg, scrtseg, sdelseg, sgetseg, ssetseg, ssetthresh

scopytable

Function

Copies tables from one database to another.

Syntax

```
scopytable [-names table_names] [-triggers]
   [-indexes] [-permissions] [-bindings] [-data]
   [-source db_path] [-target db_path]
   [-wildcard] [-version] [-help]
```

Parameters

–names *table_names* – specifies the names of the tables to copy. The names of the tables can be prefixed with the name of the owner as follows: *owner.table_name*. If you do not specify **–names**, then all tables in the specified database are copied. The following information is copied:

- table owner
- table columns
- segment

–triggers – copies all table triggers. The default is to copy the table schema, not the triggers.

–indexes – copies all table indexes. The default is to copy the table schema, not the indexes.

–permissions – copies the permissions associated with this object.

–bindings – copies the defaults and rules that are bound to this table. The default is to copy table schema, not bindings.

–data – copies table data. The default is to copy table schema, not data.

–source *db_path* – specifies the database from which to copy the tables, where *db_path* can be any valid managed database name. The default source is the current managed database. If you do not specify a source database, you must specify a target database using **–target**.

–target *db_path* – specifies the database in which to copy the tables from the source database, where *db_path* can be any valid managed database name. The default target is the current managed database. If you do not specify a target database, you must specify a source database using **–source**.

–wildcard – enables wildcard pattern matching on the table names.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scopytable -names salary_tbl -triggers -index
-data -source PARIS/devel_hrdb -target
ROME/prod_hrdb
```

This command copies the table *salary_tbl* from the source database *devel_hrdb*, in the managed SQL Server PARIS to the target database *prod_hrdb*, in the managed SQL Server ROME. In addition to the table structure, triggers, and indexes are copied.

Comments

- To use the **scopytable** command to copy table data, the SQL Server installations must be on the same management host. To copy table data to SQL Server installations that are using different management hosts, use profile distribution with data copy enabled.
- The data copy feature of Enterprise SQL Server Manager is not intended to be a replacement for Replication Server®. The process is constrained by system resources such as CPU and, more importantly, available disk space. Therefore, do not use Enterprise SQL Server Manager to copy very large tables such as data entry tables commonly used for transaction processing systems.
- Use Enterprise SQL Server Manager to copy supporting application tables such as lookup tables (state abbreviations, zip codes, and so on).
- Before you copy table data using **scopytable**, you need to be sure that you have enough disk space to complete the procedure. Before Enterprise SQL Server Manager copies a table, it renames the target table. This allows Enterprise SQL Server Manager to restore the original table if the procedure fails. Therefore, during the process, you need to have enough space in the database to hold a copy of all tables being copied. If no errors occur during the table copy, the renamed table is deleted.

Enterprise SQL Server Manager uses the Bulk Copy (BCP) utility to export and import the data from the source table. By default, the BCP files are stored in the *$DBDIR*. *$DBDIR* is the location of the Tivoli database. You must have enough disk space in *$DBDIR* to accommodate these BCP files. As a rule, make sure the *$DBDIR* has free disk space equivalent to 1.5 times the size of the tables being copied.

If *$DBDIR* is not large enough, you can use an environment variable to specify an alternate location for the BCP files. The root user must have write permission in the directory you specify. The variable is *ESSM_STAGING_DIR*. Set the variable in your own environment, such as your *.login* or *.profile* file.

Permissions

To use **scopytable**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	Database Owner of target database

See Also

sgettable, scrttable, sdeltable, ssettable, scomptable, schecktable

scopytrigger

Function

Copies triggers from one database to another. The triggers are copied to tables of the same name.

Syntax

```
scopytrigger [-names trigger_names]
   [-source db_path] [-target db_path]
   [-wildcard] [-version] [-help]
```

Parameters

–names *trigger_names* – specifies the names of the triggers to copy. The names of the triggers can be prefixed with the name of the owner as follows: owner.trigger_name. If you do not specify **–names**, all the triggers defined in the database are copied. This command copies trigger schema including:

- name
- owner
- type
- SQL

–source *db_path* – specifies the database from which to copy the triggers, where *db_path* can be any valid managed database name. The default source is the current managed database. If you do not specify a source database, you must specify a target database using **–target**.

–target *db_path* – specifies the database in which to copy the triggers from the source database, where *db_path* can be any valid managed database name. The default target is the current managed database. If you do not specify a target database, you must specify a source database using **–source**.

–wildcard – enables wildcard pattern matching on the trigger names.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scopytrigger -names delete -source
BOSTON/salesQ195 -target BOSTON/salesQ295
```

This command copies the trigger named *delete* from the source database *salesQ195*, in the managed SQL Server BOSTON to the target database *salesQ295*, in the managed SQL Server BOSTON.

Permissions

To use **scopytrigger**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	table owner in target database

See Also

sgettrigger, scrttrigger, sdeltrigger, scomptrigger

scopyuser

Function

Copies users between databases. The **scopyuser** command copies the following user attributes:

- User's login name
- Group membership
- List of login aliases (optional)

Syntax

```
scopyuser [–names user_names [–wildcard]] [–aliases]
   {[–source db_path] [–target db_path]}
   [-permissions] [–version] [–help]
```

Parameters

–names *user_names* – specifies which database users to copy, where *user_names* lists one or more database user_names in the source database. **scopyuser** copies each named user in the source database to an identical name on the target database. Without this option, **scopyuser** copies all users occurring on the source database. If the list of user_names contains spaces, you must enclose the list in double quotation marks (see "Option Lists" on page 1-3).

–wildcard – enables wildcard matching on *user_names* (see "Wildcards" on page 1-4).

–aliases – includes the list of login aliases for each user, in addition to the other user attributes being copied. **scopyuser** copies login alias information only (not the logins themselves).

–source *db_path* – specifies the database from which to copy *user_names*, where *db_path* can be any valid managed database name (see "Database Names" on page 1-16). The default is the current database collection (see "Command Context" on page 1-15). If you do not specify a source database, you must specify a target database using **–target**.

–target *db_path* – specifies the database in which to copy *user_names* from the source database, where *db_path* can be any valid managed database name (see "Database Names" on page 1-16). The default is the current database collection (see "Command Context" on page 1-15). If you do not specify a target database, you must specify a source database using **–source** (see "Comments").

–permissions – copies the command permissions associated with each user.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`scopyuser –name dave –target pubs2`**

   Copies the user *dave* from the current database to the *pubs2* database in the current managed SQL Server.

2. **`scopyuser –s BOSTON/pubs2 –t CENTRAL/newpubs`**

   Copies all users from the *pubs2* database on the managed SQL Server BOSTON to the *newpubs* database on the managed SQL Server CENTRAL.

Comments

- You must copy the user's login before copying the user. To copy SQL Server logins, use **scopylogin**.
- Before using **scopyuser**, you need the following information:
  - The usernames to copy
  - The names of the source and target databases
- The **–source** and **–target** database names must be different. If you specify the same database name for both options, **scopyuser** displays an error message.

  Because both **–source** and **–target** options default to the current database collection (see "Command Context" on page 1-15), you must supply at least one of the two database names. If you omit both options, **scopyuser** displays an error message.
- If an error occurs, **scopyuser** rolls back the entire copy operation, restoring all affected objects to their original state.
- If any of the specified usernames already exist on the target database, the **scopyuser** command fails and displays an error message.

Permissions

To use **scopyuser**, you must have the following roles:

TME	ESSM	SQL Server
any	**security**	**sa_role** or Database Owner

See Also

scompuser, scopylogin, scrtuser, sdeluser, sgetlogin, sgetuser, ssetlogin, ssetuser

scopyview

Function

Copies views from one database to another.

Syntax

```
scopyview [-names view_names] [-permissions]
   [-source db_path] [-target db_path] [-wildcard]
   [-version] [-help]
```

Parameters

–names *view_names* – specifies the names of the views to copy. The names of the views can be prefixed with the name of the owner as follows: owner.view_name. If you do not specify **–names**, all the views defined in the database are copied. The following information is copied:

- name
- owner
- column titles
- check option
- view select SQL

–permissions – copies the object permissions associated with the views you are copying.

–source *db_path* – specifies the database from which to copy the views, where *db_path* can be any valid managed database name. The default source is the current managed database. If you do not specify a source database, you must specify a target database using **–target**.

–target *db_path* – specifies the database in which to copy the views from the source database, where *db_path* can be any valid managed database name. The default target is the current managed database. If you do not specify a target database, you must specify a source database using **–source**.

–wildcard – enables wildcard pattern matching on the view names.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

➤ ***Note***

Views can depend on other views. **scopyview** copies in the proper dependency order, but it does not know if a prerequisite view is missing from the names list. You must be sure that the list includes all required views. If a prerequisite view is missing from the list, the dependent view cannot be created on the target database.

Example

```
scopyview -names small_orders
-source BOSTON/salesQ195 -target BOSTON/salesQ295
```

This example copies the view *small_orders* from the source database, *salesQ195,* in the managed SQL Server BOSTON to the target database, *salesQ295,* in the managed SQL Server BOSTON.

Permissions

To use **scopyview**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	Database Owner of the target database

See Also

sgetview, scrtview, sdelview, scompview

scrtauditrec

Function

Adds a record (comment) to a SQL Server audit database, *sybsecurity*. The **scrtauditrec** command adds the record to the audit trail (the system table, *sysaudits*).

Syntax

```
scrtauditrec
   {[-record message_text] | [-file file_name]}
   [-database db_path] [-object object_name]
   [-server server_name] [–version] [–help]
```

Parameters

–record *message_text* – specifies the text of the record to add to the audit trail, where *message_text* can be up to 255 characters of text. This text goes into the *extrainfo* field of *sysaudits*.

–file *filename* – specifies the name of the file (*filename*) containing the text of the audit record. This is useful when the message text is too long to fit on the command line. The **scrtauditrec** command takes the first line of the specified file as input, up to the first newline character. The message text can be up to 255 characters. This text goes into the *extrainfo* field of *sysaudits*.

–database *db_path* – specifies the database name that is to appear in the audit record, where *db_path* is the name of the database. This text goes into the *dbname* field of *sysaudits*. Collection path names are invalid here (use **–server** to specify a SQL Server collection object).

–object *object_name* – specifies the object name that is to appear in the audit record, where *object_name* is the name of the object. This text goes into the *objname* field of *sysaudits*.

–server *server_name* – specifies the name of the SQL Server installation that contains the audit database. The default is the current managed SQL Server collection (see "Command Context" on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scrtauditrec -rec "This is a comment."
-database pubs2 -object titles
```

Enters a user-defined audit record into the audit trail for the *titles* table in the *pubs2* database (in the current managed SQL Server).

Comments

- Before using **scrtauditrec**, you must know the name of the managed SQL Server that contains the audit database you want to access.
- Before you can use **scrtauditrec**:
  - The auditing system must be installed and running on the managed SQL Server (see **sinstallaudit**)
  - You must enable ad hoc audit records using the **–adhoc** argument to **ssetserveraudit**
- If an error occurs, **scrtauditrec** rolls back the entire create operation, restoring all affected objects to their original state.

Permissions

To use **scrtauditrec**, you must have the following roles:

TME	ESSM	SQL Server
any	**security**	**sso_role**

See Also

sgetdbaudit, sgetloginaudit, sgetprocaudit, sgetroleaudit, sgetserveraudit, sgettableaudit, sinstallaudit, ssetdbaudit, ssetloginaudit, ssetprocaudit, ssetroleaudit, ssetserveraudit, ssettableaudit

scrtcolumn

Function

Creates a column on a table within a database.

Syntax

```
scrtcolumn -name column_name -table table_name
   -type type [-length length_value]
   [-precision precision_value] [-default expression]
   [-check check_condition] [-scale scale_value]
   [-identity] [-database db_path]
   [-version] [-help]
```

Parameters

–name *column_name* – specifies the name of the new column.

–table *table_name* – specifies the name of the table to add the column to. The name of the table can be prefixed with the name of the owner as follows: owner.table_name.

–type *type* – specifies the datatype of the column.

–length *value* – specifies the length of the char, varchar, nchar, nvarchar, binary, and varbinary types. If you do not specify **–length**, and one of these types is used, the SQL Server default is 1.

–precision *value* – specifies the precision for float, numeric, and decimal types. The default value for float defaults is a platform-specific value. The default value for numeric and decimal is 18.

–scale *value* – specifies the scale value for numeric and decimal types. The SQL Server default is 0.

–default *expression* – specifies the value for the column level default. For example:

```
-default "0"
```

–check *check_condition* – specifies the name of the constraint followed by the text of the check constraint. You must specify the keyword “check” in the text of the constraint, for example:

```
-check "'my_check_constraint', check'(pub_id in
("1389", "0736"))'"
```

–identity – specifies that the column contains a system-generated, sequential value that identifies each row in the table. The default value is to not allow nulls in the column. Only one column can be specified with the **–identity** option.

–database *db_path* – specifies the database in which the table exists. The default is the current managed database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scrtcolumn -name raise_percentage -type int
-table salary_table -database HR1/FY95
```

This command creates a column called *raise_percentage* in the table *salary_table* in the database *FY95* in the managed SQL Server HR1. The column has the type *int*.

Comment

By default, **scrtcolumn** always allows null values in the column being created.

Permissions

To use **scrttable**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	Database Owner

See Also

sdelcolumn

scrtdatatype

Function

Creates a user datatype within a database.

Syntax

```
scrtdatatype -name datatype_name [-owner owner_name]
   -type physical_type
   [-length length |
       [-precision precision [-scale scale]]]
   [-identity | [-nulls | -nonulls]]
   [-bindrule rule_name] [-binddefault default_name]
   [-database db_path] [-version] [-help]
```

Parameters

–name *datatype_name* – specifies the name of the user datatype to create. Type name must conform to the rules for identifiers and must be unique for each owner in the database.

–owner *owner_name* – specifies the owner of the user datatype.

–type *physical_type* – specifies the physical or SQL Server-supplied datatype on which to base the user datatype. You can specify any SQL Server datatype except “timestamp”. Valid choices are:

floatn, float, datetimn, datetime, real, numericn, numeric, decimaln, decimal, moneyn, money, smallmoney, smalldatetime, intn, int, smallint, tinyint, bit, varchar, sysname, nvarchar, char, nchar, varbinary, binary, text, image.

–length *length* – specifies a length for the char, varchar, nchar, nvarchar, binary, and varbinary types. The SQL Server default is 1.

–precision *precision* – specifies a precision value for the float, numeric, and decimal types. The default value for float is platform-specific. The default value for numeric and decimal is 18.

–scale *scale* – specifies a scale value for the numeric and decimal types. The SQL Server default is 0.

–identity – denotes that the datatype has the IDENTITY property. IDENTITY columns are used to store sequential numbers. You can specify this option only for numeric datatypes with a scale of 0. This is not the default. If not specified, type defaults to the null mode specified by the database.

–null – allows null as a valid value. The default is the null mode specified by the database (the "allow nulls by default" option).

–nonull – do not allow null as a valid value. The default is the null mode specified by the database (the "allow nulls by default" option).

–bindrule *rule_name* – specifies the name of a rule to bind to this datatype. By default, no rule is bound to the datatype when it is created.

–binddefault *default_name* – specifies the name of the default value to bind to this datatype. By default, no default is bound to the datatype when it is created.

–database *db_path* – specifies the database in which to create the type. The default is the current managed database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scrtdatatype -name ssn -type varchar -length 11
-database PARIS/hrdb
```

This command creates a datatype called *ssn*, which is based on the *varchar* type. The datatype is created in the database *hrdb* in the managed SQL Server PARIS.

Permissions

To use **scrtdatatype**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	any

See Also

sgetdatatype, sdeldatatype, ssetdatatype, scopydatatype, scompdatatype

scrtdb

Function

Creates a new database on a specific managed SQL Server, and registers the database object in the Sybase Enterprise Management Architecture.

Syntax

```
scrtdb –name db_name [–owner owner_name]
   [–datadevices "{default | device_name } device_size
       [, device_name device_size]..." ]
   [–logdevices "{default | device_name} device_size
       [, device_name device_size]..." ]
   [–config "db_option Boolean
       [, db_option Boolean]..." ]
   [–override] [–forload]
   [–server server_name] [–version] [–help]
```

Parameters

–name *db_name* – specifies the name of the database to create, where *db_name* can be any valid database name. The name must conform to the SQL Server rules for identifiers (see the *SQL Server Reference Manual*). Collection path names are invalid here (use **–server** to specify a SQL Server collection object).

–owner *owner_name* – specifies the name of the Database Owner, where *owner_name* is an existing SQL Server login name. The default owner is “dbo”.

–datadevices – specifies the database devices (**default** or *device*) and the corresponding space (*size*) on each device to allocate to the new database for storing data. If you omit this option, **scrtdb** uses the SQL Server’s default pool of devices. You must enclose the list of *device size* pairs in double quotation marks (see “Option Lists” on page 1-3).

default – indicates that **scrtdb** can put the new database on any of the default database devices in SQL Server (listed by **sgetdev**.)

device_name – specifies the logical name of a device on which the new database can store information. A database can occupy different amounts of space on each of several database devices, thus you can specify multiple *device size* pairs in the argument list.

device_size – specifies the amount of space (in MB) to allocate on the specified device (**default** or *device*).

–logdevices – specifies the database devices (**default** or *device*) and the corresponding space (*size*) on each device to allocate to the database for storing the transaction log. You must enclose the list of *device size* pairs in double quotation marks (see “Option Lists” on page 1-3).

–config – enables or disables one or more database configuration options (*db_option*). This argument accepts a list of *db_option* {**true** | **false**} pairs. Because the argument list always contains spaces, you must enclose it in double quotation marks (see “Option Lists” on page 1-3).

db_option – specifies a database configuration option to enable or disable. The valid database option names are:

abort tran on log full
allow nulls by default
auto identity
dbo use only
ddl in tran
no chkpt on recovery
no free space acctg
read only
select into/bulkcopy
single user
trunc log on chkpt

See Appendix A, “Database Options” for a description of these database options. SQL Server accepts any unique string that is part of the option name. If you specify an option that contains spaces, you must enclose the option in single quotation marks.

Boolean – SQL Server Boolean keyword that enables or disables the preceding *db_option*. TRUE enables the option; FALSE disables the option. Because these Boolean argument values are interpreted by SQL Server as keywords, only the values TRUE and FALSE are valid (see “Boolean Keywords” on page 1-3).

–override – allows you to specify the same device name for both the data and the transaction log. This option allows SQL Server, on a machine with limited space, to still maintain the transaction log on separate device fragments from the database’s data. Without this option, the **scrtdb** command fails if you attempt to assign the data and transaction log to the same device.

–forload – creates a database for use only in loading a database dump. Use this option to recover from a media failure or to move a database from one machine to another.

–server *server_name* – specifies the managed SQL Server in which to create the database, where *server_name* can be any valid managed SQL Server name (see “SQL Server Names” on page 1-15). The default is the current managed SQL Server collection (see “Command Context” on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`scrtdb –name sampledb –config "'ddl in tran' TRUE" –server BOSTON`**

   Creates a database named *sampledb* in the managed SQL Server BOSTON. This command allocates space for the database from the Server’s default database device pool. The **–config** argument enables the database’s **ddl in tran** configuration option to allow data definition commands in transactions.

2. **`scrtdb –name newdb –datadev "default 4, dbdev1 2" –log "tranlog 2" –server BOSTON`**

   Creates a database named *newdb* in the managed SQL Server BOSTON. The **–datadevices** argument allocates 6MB of space for data divided between two separate devices—4MB on the *default* device pool and 2MB on *dbdev1*. The **–logdevices** argument allocates 2MB on *tranlog* for the transaction log. The total size of the database is 8MB.

Comments

- Before using **scrtdb**, you must know the name of the database to create. You should also decide on:
  - The names of the devices on which to place the database and transaction log (see **sgetdev**)
  - The desired size of the database and its transaction log
- The size of the database defaults to the size of the *model* database or to the size specified by the **database size** SQL Server configuration variable, whichever is larger (see **sgetserver**). The legal values for database size range from 2MB to 2^{23}MB, inclusive.

If there isn't enough space available, **scrtdb** allocates what it can and returns an appropriate message indicating how much was allocated.

- If an error occurs, **scrtdb** rolls back the entire create operation, restoring all affected objects to their original state.
- If the specified database already exists in SQL Server, the **scrtdb** command fails and displays an error message.

Permissions

To use **scrtdb**, you must have the following roles:

TME	ESSM	SQL Server
any	**space**	**sa_role**

See Also

scheckdb, scompdb, scopydb, sdeldb, sgetdb, sgetserver, ssetdb, ssetserver

scrtdefault

Function

Creates a default value within a database. This default can be bound to a column when it is created.

Syntax

```
scrtdefault -name default_name -value expression
   [-bind table_columns_or_datatypes [-future]]
   [-owner owner] [-database db_path] [-version]
   [-help]
```

Parameters

–name *default_name* – specifies the name of the default to create.

–value *expression* – specifies a constant expression representing the default value for a column. The expression cannot include the names of any columns or other database objects. You can use built-in functions that do not reference database objects. You must protect constant values with double quotes.

–owner *owner* – specifies the owner of the default. The default owner is Database Owner.

–bind – specifies a list of table columns, or user datatypes, or both to bind this default to. If you do not specify **–bind**, the default value is not bound to any column or datatype. Table column names take the format *owner.table_name.column_name*. User datatypes take the format *owner.type*.

–future – applies the default to columns with the user datatype that get created in the future. The default is not bound to existing columns. Only applies to bindings to user datatypes.

–database *db_path* – specifies the database in which to create the default. The default database is the current managed database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scrtdefault -name sex_def -value ’"F"’
-bind hr_table.sex -database PARIS/hrdb
```

This command creates a default named *sex_def* with the value “F” in the database *hrdb* in the managed SQL Server PARIS. The default is bound to the column *sex* in the table *hr_table*.

Comment

You must protect the value in the **–value** argument in double quotes.

Permissions

To use **scrtdefault**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	Database Owner. If you are specifying bindings, you must be the owner of the object being bound to.

See Also

sgetdefault, sdeldefault, ssetdefault, scopydefault, scompdefault

scrtdev

Function

Creates a new database device in a managed SQL Server.

Syntax

```
scrtdev –name device_name [–server server_name]
   –physical physical_name –size size
   [–default Boolean] [–contiguous]
   –vdevno virtual_device_num
   [–vaddr virtual_start_addr]
   [–controller disk_cont_num] [–version] [–help]
```

Parameters

–name *device_name* – specifies the logical name of the device to create. The name must conform to the SQL Server rules for identifiers (see *SQL Server Reference Manual*).

–server *server_name* – specifies the managed SQL Server in which to create *device_name*, where *server_name* can be any valid managed SQL Server name (see “SQL Server Names” on page 1-15). The default is the current managed SQL Server collection (see “Command Context” on page 1-15).

–physical *physical_name* – specifies the physical name of the new device.

–size *size* – specifies the size of the device in MB.

–default *Boolean* – specifies whether to include *device_name* in the SQL Server’s pool of default devices. TRUE includes the device; FALSE excludes the device. The default is FALSE. This option accepts any Boolean argument value (see “Boolean Keywords” on page 1-3).

–contiguous – specifies contiguous files for VMS-based SQL Servers only. If the device is a file, then this option forces the file to be created contiguously.

–vdevno *virtual_device_num* – specifies the virtual device number for the new device. Legal virtual device numbers are between 1 and 255, inclusive. This option defaults to the next available virtual device number.

–vaddr *virtual_start_addr* – specifies the virtual start address for the new device. The value for *virtual_start_addr* should be 0 (the default); reset this value only if Sybase Technical Support instructs you to do so.

–controller *disk_cont_num* – specifies the disk controller number for the new device. The default disk controller number is 0; reset this value only if Sybase Technical Support instructs you to do so.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`scrtdev –name dev1 –server UMMG –physical /dev/sd0b –size 20 –vdevno 128`**

   Creates a device named *dev1* on the SQL Server UMMG. The physical device name is */dev/sd0b* the size is 20MB, and the virtual device number is 128.

2. **`scrtdev –na defdev9 –phys /dev/sd0a -siz 15 –vdev 127 –default TRUE`**

   Creates a device named *defdev9* in the current managed SQL Server. The physical device name is */dev/sd0a*, the size is 15MB, and the virtual device number is 127. The **–default** option directs SQL Server to include *defdev9* in the pool of default devices.

Comments

- Before using **scrtdev**, you need the following information:
  - The name of the managed SQL Server in which you wish to create the device
  - The logical device name
  - The physical device name
  - The size of the device
- The **–vaddr** and **–controller** flags are advanced configuration options. You should only specify these options if Sybase Technical Support instructs you to do so.
- If an error occurs, **scrtdev** rolls back the entire create operation, restoring all affected objects to their original state.

- If the specified logical device name already exists in the SQL Server, the **scrtdev** command fails and displays an error message.
- To mirror this device, use **scrtmirror**.
- To allocate space on this device to a database, use **scrtdb** and **ssetdb**.

Permissions

To use **scrtdev**, you must have the following roles:

TME	ESSM	SQL Server
any	**space**	**sa_role**

See Also

scompdev, scopydev, scrtdb, scrtmirror, sdeldev, sgetdev, ssetdb, ssetdev

scrtdumpdev

Function

Creates a new dump device in a managed SQL Server.

Syntax

```
scrtdumpdev -name device_name [-server server_name]
   -physical physical_name
   {-disk | [[-tape] -size size]}
   [-version] [-help]
```

Parameters

–name *device_name* – specifies the logical name of the dump device to create. The name must conform to the SQL Server rules for identifiers (see the *SQL Server Reference Manual*).

–server *server_name* – specifies the name of the managed SQL Server on which the device should be created. The default is the current managed SQL Server collection (see “Command Context” on page 1-15).

–physical *physical_name* – specifies the physical name of the new device, for example */dev/rmt0*.

–disk – specifies that the dump device is a disk file. If you use this option, the **–tape** and **–size** options are invalid.

–tape – specifies that the dump device is a tape. If you use the **–tape** option, you must also specify the **–size** option.

–size *size* – specifies the size of the tape in megabytes. If you include this option, the **–tape** option is unnecessary. If you use this option, the **–disk** option is invalid.

Example

```
scrtdumpdev -name data_dev1 –physical
/devs/data_dev1 -disk -server KOKO
```

Creates a dump device *data_dev1* on the managed SQL Server KOKO. The device is a disk.

Comment

Creating a device consists of creating an entry in the *sysdevices* table that informs the SQL Server of the name and other attributes of a device. Before using **scrtdumpdev**, you must have the following information:

- The name of the managed SQL Server on which you wish to create the device.
- The logical name you want to use to refer to the device, such as tape1. This must be different from any existing dump device or database device name (see **sgetdev** and **sgetdumpdev**).
- The physical name of the actual device.

Permissions

To use **scrtdumpdev**, you must have the following roles:

TME	ESSM	SQL Server
any	**space**	**sa_role**

See Also

scompdumpdev, sgetdumpdev, scopydumpdev, ssetdumpdev

scrtgroup

Function

Creates a user group in a database.

Syntax

```
scrtgroup –name group_name [–addusers user_names]
   [–database db_path] [–version] [–help]
```

Parameters

–name *group_name* – specifies the name of the group, where *group_name* can be any valid group name. The name must conform to the SQL Server rules for identifiers (see the *SQL Server Reference Manual*).

–addusers *user_names* – specifies which users to add to the group *group_name*, where *user_names* lists one or more database user names in the database. Without this option, **scrtgroup** creates an empty group. If the list of user names contains spaces, you must enclose the list in double quotation marks (see "Option Lists" on page 1-3).

–database *db_path* – specifies the database in which to create *group_name*, where *db_path* is a valid managed database name (see "Database Names" on page 1-16). The default is the current database collection (see "Command Context" on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`scrtgroup –name trumpets –database KOKO/book`**

   Creates an empty group named *trumpets* in the *book* database in the managed Server KOKO. You can add user members to the group at a later time using **ssetgroup**, **scrtuser**, or **ssetuser.**

2. **`scrtgroup –name bones –addusers "brittw jsanders quentinj"`**

   Creates a group named *bones* in the current managed database, with database user members *brittw*, *jsanders*, and *quentinj*.

Comments

- Before using **scrtgroup**, you need the following information:
  - The name of the group to create.
  - Whether or not the users to be included in the group are already members of another group. Every user can be a member of only one group. By default, users are members of the group *public*.
- If you include the **–addusers** argument, you must make sure the specified user names exist in the database in which you are creating the new group. Before adding users to a group, use **sgetuser –onlynames** to ensure that the proposed members can access the database.
- If an error occurs, **scrtgroup** rolls back the entire create operation, restoring all affected objects to their original state.
- If the specified group name already exists on the database, the **scrtgroup** command fails and displays an error message.

Permissions

To use **scrtgroup**, you must have the following roles:

TME	ESSM	SQL Server
any	**security**	**sa_role** or Database Owner

See Also

scompgroup, scopygroup, scrtgroup, scrtuser, sdelgroup, sgetgroup, sgetuser, ssetgroup, ssetuser

scrtindex

Function

Creates an index on columns in a table.

Syntax

```
scrtindex -table table_name -name index_name
   -columns column_names [-owner owner]
   [-fillfactor percentage] [-segment segment_name]
   [-unique | -nonunique]
   [-clustered | -nonclustered] [-ignoredupkey]
   [-duprow { allow | disallow | ignore }]
   [-sorted] [-database db_path] [-version] [-help]
```

Parameters

–table *table_name* – specifies the name of the table on which to create the index.

–name *index_name* – specifies the name of the index to create. Index names must be unique within a table, but do not have to be unique within a database.

–columns *column_names* – specifies the names of the columns on which to create the index. You must specify at least one column. If you specify multiple columns, the sum of the column widths cannot exceed 256 bytes.

–owner *owner* – specifies the owner of the index. The default owner is Database Owner.

–fillfactor *percentage* – specifies how full SQL Server will make each page when it creates a new index on existing data. Valid values include 1 through 100. The default value is 0.

–segment *segment_name* – specifies the segment on which to create the index. The default is the *default* segment.

–unique – prohibits duplicate index (key) values. The default is to allow duplicate index values.

–nonunique – allows duplicate index (key) values.

–clustered – makes the physical order of the rows on this database device the same as the indexed order of the rows. A table can have only one clustered index. The default is nonclustered.

–nonclustered – specifies that there is a level of indirection between the index structure and the data itself.

–ignoredupkey – tells SQL Server to ignore attempts to enter duplicate keys into the table.

–duprow – tells SQL Server how to handle duplicate rows during index creation. If you specify **allow**, the server allows duplicate rows to exist in the table. If you specify **disallow,** the index creation fails if duplicate rows are encountered. If you specify **ignore**, SQL Server ignores duplicate rows (essentially, gets rid of them) when creating the index. This option is not relevant when creating a non-unique, nonclustered index. The default is **disallow.**

–sorted – speeds the creation of the index when the data in the table is already sorted. If the data is not already sorted, a message is issued and the index is not created.

–database *db_path* – specifies the database in which *table_name* exists. The default is the current managed database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scrtindex -name primary_key -table salary_table
-columns ssn -unique -segment public_seg1
-database PARIS/hrdb
```

This command creates an index named *primary_key* in column *ssn* of table *salary_table* in database *hrdb* in the managed SQL Server PARIS. The index is created on segment *public_seg1*. This index cannot have duplicate key values.

Permissions

To use **scrtindex**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	table owner

See Also

scheckindex, sgetindex, sdelindex, ssetindex, scopyindex, scompindex

scrtlogin

Function

Creates a new SQL Server login.

Syntax

```
scrtlogin –name login_name
   {–password password | –sybasepassword password}
   [–fullname full_name] [–defaultdatabase db_path]
   [–language default_language] [–lock Boolean]
   [–addroles roles] [–server server_name]
   [–version] [–help]
```

Parameters

–name *login_name* – specifies the login you are adding to SQL Server, where *login_name* can be any valid SQL Server login name. The name must conform to SQL Server rules for identifiers (see the *SQL Server Reference Manual*).

–password *password* – specifies the password for the new login. The *password* must be at least six characters long and can contain any printable letters, numerals, or symbols.

–sybasepassword *password* – specifies the password for the new login and also assigns the SQL Server login to the TME administrator login with the same name (*login_name*). The administrator login assignment applies to the specified SQL Server only (*server_name*).

The **–sybasepassword** option is a useful shortcut for creating a new SQL Server login and assigning it to an administrator in one command. Using this option is equivalent to executing the following command:

```
ssetsybaselogin –principal login_name –password
password –server server_name
```

–fullname *full_name* – specifies the full name of the user who owns the login. This can be useful for documentation and identification purposes. By default, **scrtlogin** does not assign a full name to the new login.

–defaultdatabase *db_path* – specifies the user’s default database. If you do not specify a database, the default is *master*.

–language *default_language* – is the official name of the default language assigned when a user logs in to SQL Server. If you do not specify *default_language*, **scrtlogin** uses the language specified by SQL Server configuration variable **default language** (see the *Enterprise SQL Server Manager User's Guide* for a description of SQL Server configuration variables).

–lock *Boolean* – indicates whether to lock the login (TRUE to lock; FALSE not to lock). The default is not to lock (FALSE). This option accepts any Boolean argument value (see "Boolean Keywords" on page 1-3).

–addroles *roles* – specifies one or more system administration roles that the new login can assume. Permissible values are:

- **sa_role** (System Administrator)
- **sso_role** (System Security Officer)
- **oper_role** (Operator)

Roles must be lowercase (see "SQL Server Keywords and Object Names" on page 1-3). By default, **scrtlogin** does not assign any roles to the new login.

–server *server_name* – specifies the managed SQL Server on which to create *login_name*, where *server_name* can be any valid managed SQL Server name (see "SQL Server Names" on page 1-15). The default is the current managed SQL Server collection (see "Command Context" on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`scrtlogin –name duke –password CottonTail`**

   Creates the user login *duke* in the current managed SQL Server, and assigns the password *CottonTail* to the login. **scrtlogin** assigns default values for all other unspecified command line options.

2. **`scrtlogin –n paulg –p 27chorus –full "Paul Gonzalves" -serv NEWPORT –data pubs2`**

   Creates the user login *paulg* with the password *27chorus* and full name *Paul Gonzalves* on the SQL Server named NEWPORT, and assigns default database *pubs2* to the login. **scrtlogin** assigns default values for all other unspecified command line options.

3. **`scrtlogin –n billys –p BloodCount –addroles "sa_role sso_role" -s UMMG`**

   Creates the login *billys* with the password *BloodCount* on the SQL Server UMMG, and assigns the System Administrator (**sa_role**) and System Security Officer (**sso_role**) roles to the login. **scrtlogin** assigns default values for all other unspecified command line options.

Comments

- Before using **scrtlogin**, you need the following information:
  - A unique name for the login
  - The password to assign the login
- To simplify system management and eliminate confusion, it is usually best to make users' SQL Server login names match their operating system login names.
- This command removes all password information from the command line to prevent unauthorized users from viewing the password via the UNIX **ps** command.
- If an error occurs, **scrtlogin** rolls back the entire create operation, restoring all affected objects to their original state.
- If the specified login name already exists on SQL Server, the **scrtlogin** command fails and displays an error message.

Permissions

To use **scrtlogin**, you must have the following roles:

TME	ESSM	SQL Server
any. You must have **super** role to use the **–sybasepassword** option	**security**	**sso_role**

See Also

scomplogin, scopylogin, scrtsybaselogin, sdellogin, sgetlogin, sgetloginaudit, ssetlogin, ssetloginaudit, ssetsybaselogin

scrtmirror

Function

Creates a mirror device for an existing device that immediately takes over when the primary device fails. You can mirror the master device, devices that store data, and devices that store transaction logs; you cannot mirror dump devices.

Syntax

```
scrtmirror –device device_name
   –physical mirror_physical_name [–disablemirror]
   [–serial Boolean] [–server server_name]
   [–version] [–help]
```

Parameters

–device *device_name* – specifies the logical name of the device to mirror.

–physical *mirror_physical_name* – specifies the physical name of the mirror device on which to mirror *device_name*.

–disablemirror – disables the mirror after it is created. This is useful if you want to create the mirror now, but enable it later using **ssetdev**.

–serial *Boolean* – specifies whether to enforce serial writes to the primary and mirror devices. Setting this option to TRUE (the default), guarantees that each write operation to the primary database device finishes before the corresponding write to the mirror device begins. If the primary and mirror devices are on different physical devices, serial writes can ensure that at least one of the disks may be unaffected in the event of a power failure. If FALSE, both I/O requests are queued immediately, one to each device. This option accepts any Boolean argument value (see “Boolean Keywords” on page 1-3).

–server *server_name* – specifies the managed SQL Server on which *device_name* exists, where *server_name* can be any valid managed SQL Server name (see “SQL Server Names” on page 1-15). The default is the current managed SQL Server collection (see “Command Context” on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scrtmirror –server WASHINGTON –device d_master
–physical /dev/sd0a –serial false
```

Creates a mirror for the master device, *d_master*, on the managed SQL Server WASHINGTON. The physical device name for the mirror is */dev/sd0a*. Writes to the primary and mirror device will occur using non-serial I/O.

Comments

- Before using **scrtmirror**, you need the following information:
  - The logical name of the primary device
  - The physical name of the mirror device
- Use **ssetdev** to enable or disable the primary and mirror devices.
- If an error occurs, **scrtmirror** rolls back the entire create operation, restoring all affected objects to their original state.

Permissions

To use **scrtmirror**, you must have the following roles:

TME	ESSM	SQL Server
any	**space**	**sa_role**

See Also

scrtdev, scompdev, scopydev, sdeldev, sgetdev, ssetdev

scrtprfmgr

Function

Creates an ESSM profile manager and associates it with either a SQL Server or a database. The profile manager manages ESSM profiles.

Syntax

```
scrtprfmgr -name name -source server_or_db_path
   [-version] [-help]
```

Parameters

–name *name* – specifies the profile manager name

–source *server_or_db_path* – specifies the managed SQL Server or database to be associated with the profile manager. This SQL Server or database is the source of all profile information when you distribute the profile information to subscribers.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scrtprfmgr -name server1_pm -source BOSTON
```

Creates a SQL Server profile manager called *server1_pm* that is associated with a managed SQL Server called BOSTON.

Comments

- You can add profiles to an ESSM profile manager using the Tivoli **wcrtprf** command.
- SQL Server profiles cannot be created within database profile managers, and database profiles cannot be created within SQL Server profile managers.
- Non-ESSM profiles cannot be added to ESSM profile managers.
- You can use the Tivoli **wsub** command to subscribe resources to a profile manager.
- Although a database cannot subscribe to a SQL Server profile manager, a SQL Server can subscribe to a database profile manager.

- If a SQL Server subscribes to a SQL database profile manager and the profile manager is distributed to the subscriber, the database is distributed to SQL Server.

Permissions

To use **scrtprfmgr**, you must have the following roles:

TME	ESSM	SQL Server
senior or **super**	none	none

See Also

sgetprf, spopulateprf

scrtproc

Function

Creates a stored procedure within a database.

Syntax

```
scrtproc -name procedure_name  [-owner owner]
   [-parameters parameter_definitions_file]
   [-recompile] [-database db_path] [-version] [-help]
```

Parameters

The SQL statements of the stored procedure must be specified via standard input (stdin). This SQL is the text that follows the **as** in the **create procedure** SQL statement.

–name *procedure_name* – specifies the name of the stored procedure to create. The name must conform to the rules for identifiers.

–owner *owner* – specifies the owner of the stored procedure. The default owner is Database Owner.

–parameters – specifies a file containing the parameters to this stored procedure. If you do not specify **–parameters**, the stored procedure will have no parameters. Specify parameters using the following options:

```
–pname name –ptype type [–plength length]
[–pprecision value] [–pscale value]
[–pdefault value] [–poutput]
```

The parameters must be specified in the order listed.

–pname *name* – defines the name of the parameter. The parameter name must begin with an “@” character. The total length of the name, including the “@”, cannot exceed 30 characters.

–ptype *type* – specifies the type of the parameter.

–plength *length* – specifies the length for the *char, varchar, nchar, nvarchar, binary,* and *varbinary* types. If you do not specify **–plength**, and you use one of these types, the default is 1.

–pprecision *value* – specifies the precision value for the float, numeric, and decimal types. The default for float is platform-specific. The default for numeric and decimal is 18.

–pscale *value* – specifies the scale value for numeric and decimal types. The default scale value is 0.

–pdefault *value* – defines the default parameter value used if the caller of this stored procedure does not specify a value.

–poutput – identifies the parameter as an output-only parameter. The default is the input parameter.

–recompile – causes SQL Server to recompile this stored procedure each time it is executed.

–database *db_path* – specifies the database in which the stored procedure will be created. The default is the current managed database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scrtproc -name locked_accounts
  -parameters proc.params
  -database PARIS_DEV/master <Return>
select name, status from syslogins
where status=@svalue and name like @name <Return>
[CTRL_D]
```

The file *proc.params* contains the following:

```
-pname @name
-ptype varchar
-plength 30
-pname @svalue
-ptype int
```

This command creates a procedure called *locked_accounts* in the database *master* in the managed SQL Server PARIS_DEV. The parameters are in the file *proc.params*.

Comments

If the file specified by **–parameters** is in the current directory, you just need to specify the file name. If the file is in a different directory, you must specify a full pathname.

Permissions

To use **scrtproc**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	Database Owner

See Also

sgetproc, sdelproc, scopyproc, scompproc

scrtrmtlogin

Function

Creates entries in a local SQL Server's *sysremotelogins* table to enable logins from a remote SQL Server, mapping the remote login names to local login names.

Syntax

```
scrtrmtlogin –names remote_login_names
   –remoteserver remote_server_name
   –locallogin login_name [–trusted Boolean]
   [–server server_name] [–version] [–help]
```

Parameters

–names *remote_login_names* – specifies which remote logins to add to the local SQL Server, where *remote_login_names* lists one or more login names existing on the remote SQL Server. **scrtrmtlogin** adds each named login on the remote SQL Server to the local SQL Server's *sysremotelogins* table. If the list of remote login names contains spaces, you must enclose the list in double quotation marks (see "Option Lists" on page 1-3).

–remoteserver *remote_server_name* – specifies the name of the remote SQL Server containing the specified remote logins. The remote SQL Server does not necessarily have to be a managed SQL Server (for more on managed servers, see **smanageserver**.

–locallogin *login_name* – maps the names in *remote_login_names* to a single login name on the local SQL Server, where *login_name* specifies an existing login on the local SQL Server.

–trusted *Boolean* – sets **trusted** or **untrusted** mode for the remote logins on the local SQL Server. FALSE specifies **untrusted** mode, and the local SQL Server performs password checking for the remote logins named in *remote_login_names*; TRUE specifies **trusted** mode, and SQL Server does not perform password checking for *remote_login_names*. The default is **untrusted** mode (FALSE). This option accepts any Boolean argument value (see "Boolean Keywords" on page 1-3).

–server *server_name* – specifies the local SQL Server in which to add the remote logins, where *server_name* can be any valid managed SQL Server name (see "SQL Server Names" on page 1-15). The default is the current managed SQL Server collection (see "Command Context" on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`scrtrmtlogin –server HARLEM –remote EAST_ST_LOUIS –name harryc –local bari`**

   Adds the user login *harryc* as a remote login from the remote SQL Server EAST_ST_LOUIS to the local SQL Server UMMG. The remote login maps to the local SQL Server login *bari*.

2. **`scrtrmtlogin –remote EAST_ST_LOUIS –name "jhodges jimmyh" –local reeds`**

   Adds the user logins *jhodges* and *jimmyh* as remote logins from remote SQL Server EAST_ST_LOUIS to the (local) current managed SQL Server. Both remote logins will be known by the local SQL Server as *reeds*.

Comments

- Before using **scrtrmtlogin**, you need the following information:
  - The name of the remote SQL Server from which the remote login may access the local SQL Server.
  - The default local login name mapping defined for the local-remote SQL Server pair. You can obtain this information by using the **sgetrmtserver** command.
  - Whether the local SQL Server can accept remote logins. This is determined by the **remote access** configuration variable. You can view the status of **remote access** for the local SQL Server using the **sgetserver** command.
- If an error occurs, **scrtrmtlogin** rolls back the entire create operation, restoring all affected objects to their original state.
- If any of the specified remote login names already exist on the local SQL Server, the **scrtrmtlogin** command fails and displays an error message.

Permissions

To use **scrtrmtlogin**, you must have the following roles:

TME	ESSM	SQL Server
any	**security**	**sa_role**

See Also

scrtrmtserver, sdelrmtlogin, sdelrmtserver, sgetrmtserver, ssetrmtlogin, ssetrmtserver

scrtrmtserver

Function

Creates a remote SQL Server entry in a local managed SQL Server's *sysservers* table so that the remote SQL Server may initiate remote procedure calls (RPCs) on the local SQL Server.

Syntax

```
scrtrmtserver –name remote_server_name
   [–timeout Boolean] [–encrypt Boolean]
   [[–locallogin login_name] | –remotelogin]
   [–server server_name] [–version] [–help]
```

Parameters

–name *remote_server_name* – specifies the remote SQL Server for which the local SQL Server is to allow access, where *remote_server_name* is the name of an existing SQL Server. The remote SQL Server does not necessarily have to be a managed SQL Server (for more on managed servers, see **smanageserver**).

–timeout *Boolean* – enables or disables the normal timeout capability used by the local SQL Server. TRUE specifies that the local SQL Server should drop the physical network connection to the remote SQL Server after one minute of inactivity. FALSE specifies that the local SQL Server should disable the automatic timeout; SQL Server will not automatically drop the connection. The default is TRUE (enable timeout capability). This option accepts any Boolean argument value (see “Boolean Keywords” on page 1-3).

–encrypt *Boolean* – enables or disables client-side, password encryption handshakes for remote SQL Server connections to the local SQL Server. TRUE enables password encryption handshakes; FALSE disables password encryption handshakes. The default is FALSE (no network password encryption). This option accepts any Boolean argument value (see “Boolean Keywords” on page 1-3).

–locallogin *login_name* – specifies that all remote logins map to a single local login, where *login_name* is an existing login on the local SQL Server. Without this option, you must individually map remote logins to local login names. (See **scrtrmtlogin** and **ssetrmtlogin**.)

–remotelogin – specifies that all logins from the remote SQL Server use their own names in connecting to the local SQL Server. Without this option, you must individually map remote logins to local login names.

–server *server_name* – specifies the local SQL Server in which to add the remote SQL Server access, where *server_name* can be any valid managed SQL Server name (see "SQL Server Names" on page 1-15). The default is the current managed SQL Server collection (see "Command Context" on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`scrtrmtserver –server UMMG –name EAST_ST_LOUIS`**

   Enters EAST_ST_LOUIS as a remote SQL Server in the *sysservers* system table of the local managed SQL Server UMMG. All default options apply. The SQL Server EAST_ST_LOUIS can now initiate remote procedure calls on the local Server UMMG.

2. **`scrtrmtserver –ser UMMG –name EAST_ST_LOUIS –encrypt true`**

   Enters EAST_ST_LOUIS as a remote SQL Server in the *sysservers* system table of the local managed SQL Server UMMG. Sets the option to encrypt passwords for network transmission.

Comments

- Before using **scrtrmtserver**, you must know the names of the local and remote servers.
- If an error occurs, **scrtrmtserver** rolls back the entire create operation, restoring all affected objects to their original state.
- If the specified remote SQL Server already exists on the local SQL Server, the **scrtrmtserver** command fails and displays an error message.

Permissions

To use **scrtrmtserver**, you must have the following roles:

TME	ESSM	SQL Server
any	**security**	**sso_role**

See Also

scrtrmtlogin, sdelrmtlogin, sdelrmtserver, sgetrmtserver, ssetrmtlogin, ssetrmtserver

scrtrule

Function

Creates a rule in a database. This rule can be bound to a column or user datatype when it is created.

Syntax

```
scrtrule -name rule_name  [-owner owner]
   [-database db_path] [-wildcard] [-version] [-help]
```

Parameters

Specify the rule definition using standard input. This value must be an expression that is valid in a "where" clause. It cannot reference any column or other database object.

–name *rule_name* – specifies the name of the rule to create.

–owner *owner* – specifies the owner of the rule. The default owner is Database Owner.

–database *db_path* – specifies the database in which to create the rule. The default is the current managed database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scrtrule -name quiz_answer
  -database PARIS_DEV/master <Return>
answer in ("A", "B", "C") <Return>
<CTRL–D>
```

This command creates a rule named *quiz_answer* in the database *master* in the managed SQL Server PARIS_DEV. The definition of the rule is **@answer in ("A", "B", "C")**.

Permissions

To use **scrtrule**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	Database Owner. If you specify **–bind**, you must be the owner of the object being bound to, you must be a user, or you must be in a group with "create rule" permission.

See Also

sgetrule, sdelrule, ssetrule, scopyrule, scomprule

scrtseg

Function

Creates a segment in a database. The segment represents one or more of the database's allocated devices.

Syntax

```
scrtseg –name segment_name –devices device_names
   [–database db_path] [–version] [–help]
```

Parameters

–name *segment_name* – specifies the name of the segment to create, where *segment_name* is the name of the segment to add to the database's *syssegments* table. The segment name must conform to SQL Server rules for identifiers (see the *SQL Server Reference Manual*), and must be unique within the database.

–devices *device_names* – specifies the logical names of the devices that comprise the segment *segment_name*. You must supply this argument.

–database *db_path* – specifies the database in which to define the segment, where *db_path* can be any valid managed database name (see "Database Names" on page 1-16). The default is the current database collection (see "Command Context" on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. `scrtseg –name index_seg –devices "dev1 dev2"`

   Creates a segment named *index_seg* composed of the database devices *dev1* and *dev2* in the current managed database.

2.
   ```
   scrtseg –name db_seg1 –devices "dev1 dev2 dev8"
   –database UMMG/pubs2
   ```

   Creates a segment named *db_seg1* composed of the database devices *dev1*, *dev2*, and *dev8*, in the *UMMG/pubs2* managed database.

Comments

- Before using **scrtseg**, you need the following information:
  - The name of the segment to create
  - The names of the database devices that make up the segment
- The database devices must exist and be available to the database. You can create the database device with **scrtdev** and make it available to the database using **scrtdb** or **ssetdb**.
- If an error occurs, **scrtseg** rolls back the entire create operation, restoring all affected objects to their original state.
- If a segment name already exists in the database, the **scrtseg** command fails and displays an error message.
- To modify segment attributes, use the **ssetseg** command.

Permissions

To use **scrtseg**, you must have the following roles:

TME	ESSM	SQL Server
any	**space**	**sa_role**

See Also

scompseg, scopyseg, scrtdb, scrtdev, sdelseg, sgetseg, ssetdb, ssetseg

scrtsybaselogin

Function

Defines a Sybase SQL Server login for an administrator. A login can be defined for the Tivoli Management Region, for a policy region, or for a particular managed SQL Server.

Syntax

scrtsybaselogin –principal *username*
   –password *password* [–name *SQLServer_login_name*]
   [[–policyregion *policyregion*] | –server *server*]
   [–version] [–help]

Parameters

–principal *username* – specifies the Tivoli name for the user. On UNIX systems, this is the UNIX login name.

–password *password* – specifies the Sybase SQL Server password for the principal username.

–name *SQLServer_login_name* – specifies the SQL Server login name. If not specified, ESSM uses the principal username.

–policyregion *policyregion* – specifies the specific policy region for which the password and SQL Server login name will be set.

–server *server* – specifies the managed SQL Server for which the password and SQL Server login name will be set.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
ssetsybaselogin -principal webster -password
bwebster -name retsbew
```

Sets the SQL Server login *retsbew* for the ESSM administrator at the Tivoli Management Region level.

Comments

- If both the **–policyregion** and the **–server** options are omitted, the password is set at the Tivoli Management Region level.

- If you create a subregion in a policy region, an administrator must have a Sybase SQL Server login for that subregion in order to access it. The Sybase SQL Server login for the parent policy region will not enable access to a subregion.

Permissions

To use **scrtsybaselogin**, you must have the following roles:

TME	ESSM	SQL Server
super	any	none

See Also

scrtlogin, ssetsybaselogin, sgetsybaselogin, sdelsybaselogin

scrttable

Function

Creates a one-column table within a database.

Syntax

```
scrttable -name table_name [-owner owner]
   -column column_name  -type type
   [-length length_value] [-precision precision_value]
   [-scale scale_value]
   [-nulls | -nonulls | -identity]
   [-check "constraint_name, 'constraint_text'"]
   [-default expression] [-segment segment_name]
   [-database db_path] [-version] [-help]
```

Parameters

–name *table_name* – specifies the name of the table to create. If the name is prefixed with a "#", then the table is a temporary table created in tempdb.

–owner *owner* – specifies the owner of the table. The default owner is Database Owner.

–column *column_name* – specifies the name of the column.

–type *type* – specifies the type of the column.

–length *length_value* – specifies length of char, varchar, nchar, nvarchar, binary, and varbinary types. If you do not specify **–length**, and you use one of the listed types, the default is 1.

–precision *precision_value* – specifies the precision for float, numeric, and decimal types. Float defaults to a SQL Server platform-specific value, numeric and decimal default to 18.

–scale *scale_value* – specifies the scale value for numeric and decimal types. The default is 0.

–nulls – specifies that the column allows null values. The default value is to not allow nulls in the column.

–nonulls – specifies that the column does not allow null values. **–nonulls** is the default value.

–default *expression* – specifies the value for the column level default. The expression must include the word “default.” For example:

```
-default ’default 0’
```

–check *“constraint_name, ‘constraint_text’”* – specifies the name of the constraint followed by the text of the check constraint. You must specify the keyword “check” in the text of the constraint. For example:

```
-check "my_check_constraint, ’check (pub_id in
("1389", "0736"))’"
```

–identity – specifies that the column contains a system-generated, sequential value that identifies each row in the table. The default value is to not allow nulls in the column. Only one column can be specified with the **–identity** option.

–segment *segment_name* – specifies the name of the segment on which to place the table.

–database *db_path* – specifies the database in which to create the table. The default is the current managed database. If the database is "tempdb," the table created is a globally accessible, temporary table.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

➤ ***Note***

This command creates a one-column table. To add additional columns to the table, use the **scrtcolumn** command.

Example

```
scrttable -name salary_table -column ssn
-type char -length 11 -database HR1/FY95
```

This command creates a table named *salary_table* in the database *FY95*, in the managed SQL Server HR1. It has a column named *ssn*, of type *char,* with a length of 11.

Permissions

To use **scrttable**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	Database Owner

See Also

sgettable, sdeltable, ssettable, scopytable, scomptable, schecktable

scrtthresh

Function

Creates a threshold to monitor space usage on a database segment. When free space on the segment falls below the specified threshold, SQL Server executes the associated stored procedure.

Syntax

```
scrtthresh –name "segment_name free_pages"
   –sproc sproc_name [–database db_path]
   [–version] [–help]
```

Parameters

–name – specifies the segment name to monitor (*segment_name*) and free page threshold (*free_pages*). You must enclose the *segment_name* and *free_pages* pair in double quotation marks (see “Option Lists” on page 1-3).

segment_name – specifies the name of an existing segment to monitor.

free_pages – specifies the threshold limit, where *free_pages* is the number of pages at which to set the threshold. When the available space on *segment_name* drops below this limit, SQL Server executes the associated stored procedure.

–sproc *sproc_name* – specifies the name of the stored procedure to execute when the available space on *segment_name* drops below the threshold.

The procedure can be on any database in the current managed SQL Server or on an Open Server. Thresholds cannot execute procedures on remote SQL Servers.

–database *db_path* – specifies the database in which the segment to be monitored (*segment_name*) exists, where *db_path* is a valid managed database name (see “Database Names” on page 1-16). The default is the current database collection (see “Command Context” on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scrtthresh –name "default 200" –database pubs2
–sproc def_threshold
```

Creates a threshold on the *default* segment in the *pubs2* database. SQL Server will execute the stored procedure **def_threshold** when the number of available pages in the *default* segment drops below 200 pages.

Comments

- Before using **scrtthresh**, you need the following information:
  - The name of the database containing the segment to monitor
  - The name of the segment to monitor
  - The page limit at which to set the threshold
  - The name of the stored procedure to execute on reaching the page limit threshold
- The stored procedure does not have to exist at the time you create the threshold.
- If an error occurs, **scrtthresh** rolls back the entire create operation, restoring all affected objects to their original state.

Permissions

To use **scrtthresh**, you must have the following roles:

TME	ESSM	SQL Server
any	**space**	**sa_role**

See Also

scompseg, scopyseg, sdelthresh, sgetseg, ssetthresh

scrttrigger

Function

Creates a trigger on a table.

Syntax

```
scrttrigger -name trigger_name -table table_name
   [-owner owner] {[-insert] [-update] [-delete]}
   [-database db_path] [-version] [-help]
```

Parameters

The SQL body of the trigger must be specified via stdin.

–table *table_name* – specifies the name of the table to create the trigger on.

–name *trigger_name* – specifies the name of the trigger to create.

–owner *owner* – specifies the owner of the trigger. The default owner is Database Owner.

–insert, **–update**, **–delete** – specifies the type of data modification that causes this trigger to fire. You must specify at least one of these parameters.

–database *db_path* – specifies the database in which the triggers exist. The default is the current managed database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scrttrigger -name order_delete -table orders
  -delete -database PARIS/FY95 <Return>
print "Order removed from orders db!" <Return>
[CTRL_D]
```

This command creates a trigger named *order_delete* in the table *orders*, in the database *FY95* in the managed SQL Server PARIS. This trigger gets fired when data is deleted from the table. It triggers printing of the text:

```
"Order removed from orders db!"
```

Permissions

To use **scrttrigger**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	table owner

See Also

sgettrigger, sdeltrigger, scopytrigger, scomptrigger

scrtuser

Function

Creates a user in a database so that the associated login has access to the database.

Syntax

```
scrtuser
   –name user_name [–login login_name]
   [–group group_name] [–addaliases login_names]
   [–database db_path] [–version] [–help]
```

Parameters

–name *user_name* – specifies the database username (the name by which the login is to be known in the database). The name must conform to SQL Server rules for identifiers (see the *SQL Server Reference Manual*). If you omit this option, you must specify a login with **–login**, and the new username defaults to *login_name*.

–login *login_name* – specifies the name of the login to be given database access. If you omit this option, you must specify a username with **–name**, and the *login_name* defaults to *user_name*.

–group *group_name* – specifies a group to which you want to add the new user, where *group_name* is the name of an existing group in the database. If you omit this option, **scrtuser** adds the new user to the default group *public*.

–addaliases *login_names* – specifies the login names for which the new *name_in_db* is an alias, where *login_names* is one or more existing logins on SQL Server. When using the database, login_names assume the identity and privileges of *name_in_db*. The login names must not already be current users in the database. If the list of logins contains spaces, you must enclose the list in double quotation marks (see “Option Lists” on page 1-3).

–database *db_path* – specifies the database in which to create *name_in_db*, where *db_path* is a valid managed database name (see “Database Names” on page 1-16). The default is the current database collection (see “Command Context” on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`scrtuser –name duke –database master`**

   Creates the database user *duke* in the *master* database. Because the login name is unspecified, the associated login name is the same as the specified username. The SQL Server login must be created before the user can be created.

2. **`scrtuser –login rnance`**

   Creates the database user *rnance* in the current managed database. Because the username is unspecified, the resulting database username is the same as the SQL Server login name. (The SQL Server login must be created before the user can be created.)

3. **`scrtuser –login bubber –name "bmiley" –database pubs2`**

   Creates the database user *bubber* for the SQL Server login *bmiley* in the *pubs2* database. (The SQL Server login must be created before the user can be created.)

4. **`scrtuser –login cat –name screech -addaliases "bubber, cootie" –database pubs2`**

   Creates the database user *screech* in the *pubs2* database for the SQL Server login *cat*. Specifies that the logins *bubber* and *cootie* can also be used to access the *pubs2* database. (The SQL Server logins must be created before the user can be created or the aliases added.)

Comments

- To use **scrtuser**, you must specify either the login name or username or both. If the database username is to be the same as the login name, you can specify either the login name or the username alone. If the user is to be known in the database by a name other than the login name, you must specify both the login name and the username explicitly.
- A login mapped to a username or alias in a database cannot be mapped to another alias in the same database. A login must be known by a single name in any database.
- If an error occurs, **scrtuser** rolls back the entire create operation, restoring all affected objects to their original state.

- If the specified user already exists on SQL Server, the **scrtuser** command fails and displays an error message.

Permissions

To use **scrtuser**, you must have the following roles:

TME	ESSM	SQL Server
any	**security**	**sa_role** or Database Owner

See Also

scompuser, scopyuser, sdeluser, ssetgroup, ssetuser

scrtview

Function

Creates a view within a database.

Syntax

```
scrtview -name view_name [-owner owner]
   [-columns column_names] [-check] [-distinct]
   [-database db_path] [-version] [-help]
```

Parameters

The select statement that defines the view must be specified via stdin. The format of this SQL statement is the body of the select statement that follows "select [distinct]" in the Transact-SQL create view statement.

–name *view_name* – name of the view to create. The name cannot include the database name.

–owner *owner* – specifies the owner of the view. The default owner is Database Owner.

–columns *column_names* – specifies the names to be used as headings for the columns in the view. If you do not specify **–columns**, the column names default to the column names specified in the select statement.

-check – specifies that all data modifications are validated against the view selection criteria. All rows inserted or updated through the view must remain visible through the view.

–database *db_path* – specifies the database in which to create the view. The default is the current managed database.

–distinct – specifies that the view cannot contain duplicate rows.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
scrtview -name price_list
  -database BOSTON/pubs2 <Return>
price from titles <Return>
[CTRL_D]
```

This command creates a view named *price_list* in the database *pubs2* in the managed SQL Server BOSTON.

Permissions

To use **scrtview**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	Database Owner

See Also

sgetview, sdelview, scopyview, scompview

sdeldatatype

Function

Deletes user-defined datatypes from within a database.

Syntax

```
sdeldatatype -names datatype_names
   [-database db_path] [-version] [-help]
```

Parameters

–names *datatype_names* – specifies the names of the datatypes to delete. The format is *owner.type_name*.

–database *db_path* – specifies the database where the datatypes reside. The default is the current managed database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
   sdeldatatype -names ssn -database PARIS/hrdb
```

This command deletes the user datatype *ssn* from the database *hrdb* in the managed SQL Server PARIS.

Permissions

To use **sdeldatatype**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	Database Owner or datatype owner

See Also

sgetdatatype, scrtdatatype, ssetdatatype, scopydatatype, scompdatatype

sdeldb

Function

Deletes a database from a managed SQL Server, including all objects in the database, and frees the allocated storage space.

Syntax

```
sdeldb –name db_name [–damaged] [–server server_name]
   [–version] [–help]
```

Parameters

–name *db_name*– specifies the database to delete, where *db_name* is the name of an existing database. Collection path names are invalid here (use **–server** to specify a SQL Server collection object).

–damaged – deletes a damaged database. Use this option to delete a database that cannot be recovered or used.

–server *server_name* – specifies the managed SQL Server from which to delete the database, where *server_name* can be any valid managed SQL Server name (see “SQL Server Names” on page 1-15). The default is the current managed SQL Server collection (see “Command Context” on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`sdeldb –name oldpubs`**

   Deletes the database named *oldpubs* from the current managed SQL Server.

2. **`sdeldb –name oldpubs –damaged –server KOKO`**

   Deletes the damaged database named *oldpubs* from the managed SQL Server KOKO.

Comments

- There is no undo for **sdeldb**.
- Before using **sdeldb**, you need the following information:
  - The name of the database to delete
  - Whether the database is damaged (see **scheckdb**)
- If anyone is using the database when you try to delete it, the **sdeldb** command fails and displays an error message.
- You cannot delete the *sybsecurity* database if auditing is currently enabled. To disable auditing, use **ssetserveraudit.**

Permissions

To use **sdeldb**, you must have the following roles:

TME	ESSM	SQL Server
any	**space**	**sa_role** or Database Owner

See Also

scheckdb, scompdb, scopydb, scrtdb, sgetdb, ssetdb, ssetserveraudit

sdeldefault

Function

Removes defaults from a database.

Syntax

```
sdeldefault -names default_names
   [-database db_path] [-version] [-help]
```

Parameters

–names *default_names* – specifies the names of the defaults to delete. The name of the defaults can be prefixed with the name of the owner as follows: owner.default_name.

–database *db_path* – specifies the database in which the defaults exist. The default database is the current managed database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
sdeldefault -names sex_def -database PARIS/hrdb
```

This command deletes the default named *sex_def* that is located in the database *hrdb* in the managed SQL Server PARIS.

Permissions

To use **sdeldefault**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	Database Owner. If you are specifying bindings, you must be the owner of the object being bound to.

See Also

sgetdefault, scrtdefault, ssetdefault, scopydefault, scompdefault

sdeldev

Function

Deletes a database device from a managed SQL Server.

Syntax

```
sdeldev –names device_names [–delfile]
   [–server server_name] [–version] [–help]
```

Parameters

–names *device_names* – specifies the logical names of the devices to delete, where *device_names* are the names of one or more existing devices.

–delfile – deletes any file system space allocated to *device_name*.

–server *server_name* – specifies the managed SQL Server from which to delete *device_name*, where *server_name* can be any valid managed SQL Server name (see “SQL Server Names” on page 1-15). The default is the current managed SQL Server collection (see “Command Context” on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
sdeldev –name dev1 –server KOKO
```

Deletes the device named *dev1* from the managed SQL Server KOKO.

Comments

- Before using **sdeldev**, you need the following information:
  - The logical names of the devices to delete
  - The name of the managed SQL Server from which to delete the database devices
- If databases are using space on a device when you try to delete it, the **sdeldev** command fails and displays an error message.
- If the SQL Server installation from which you are deleting a device is not located on the management host, when you use the **–delfile** argument, the file system space is not deleted.

- You must restart SQL Server after dropping a device in order to reclaim the associated virtual device number.

Permissions

To use **sdeldev**, you must have the following roles:

TME	**ESSM**	**SQL Server**
any	**space**	**sa_role**

See Also

scompdev, scopydev, scrtdev, scrtmirror, sgetdev, ssetdev

sdeldumpdev

Function

Deletes a dump device from a managed SQL Server.

Syntax

```
sdeldumpdev -names device_names [-server server_name]
   [-version] [-help]
```

Parameters

–names *device_names* – specifies the logical names of the devices to delete, where *device_names* is the name of one or more existing dump device.

–server *server_name* – specifies the SQL Server installation from which to delete the device. The default is the current managed SQL Server collection (see “Command Context” on page 1-15).

–version – prints the release and copyright on the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
sdeldumpdev -names tapedump1 -server AGRA
```

Deletes the dump device *tapedump1* on the managed SQL Server AGRA.

Comments

- Before using **sdeldumpdev**, you must have the following information:
  - The logical names of the dump devices to delete
  - The name of the managed SQL Server from which to delete the dump devices
- You must restart SQL Server after dropping devices in order to reclaim the virtual device numbers associated with the devices. If you don't need the numbers immediately, and you restart SQL Server regularly, you can postpone this operation.

Permissions

To use **sdeldumpdev**, you must have the following roles:

TME	ESSM	SQL Server
any	**space**	**sa_role**

See Also

sgetdumpdev, scompdumpdev, scrtdumpdev, scopydumpdev

sdelgroup

Function

Deletes a group from a database.

Syntax

```
sdelgroup –names group_names [–database db_path]
   [–version] [–help]
```

Parameters

–names *group_names* – specifies the names of the groups to delete, where *group_names* is the name of one or more existing groups.

–database *db_path* – specifies the database from which to delete *group_name*, where *db_path* is a valid managed database name (see “Database Names” on page 1-16). The default is the current database collection (see “Command Context” on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
sdelgroup –names altos –database KOKO/inventory
```

Deletes the group *altos* from the *inventory* database on the managed SQL Server KOKO.

Comments

- Before using **sdelgroup**, you need the following information:
  - The names of the groups to delete
  - The name of the database from which to delete the groups
- If an error occurs, **sdelgroup** rolls back the entire delete operation, restoring all affected objects to their original state.
- You cannot delete a group if it is the group named *public*. The **sdelgroup** command fails and displays an error message.
- When you delete a group all members are assigned to the *public* group.

Permissions

To use **sdelgroup**, you must have the following roles:

TME	ESSM	SQL Server
any	**security**	**sa_role** or Database Owner

See Also

scompgroup, scopygroup, scrtgroup, sgetgroup, ssetgroup

sdelindex

Function

Deletes indexes from a table.

Syntax

```
sdelindex -names index_names [-database db_path]
   [-version] [-help]
```

Parameters

–names *index_names* – specifies the names of the indexes to delete in the format table.indexname.

–database *db_path* – specifies the database in which the table exists. The default is the current managed database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

➤ ***Note***

This command does not allow you to delete table indexes that were created with the **create table** SQL command. Indexes of this type must be deleted using the **alter table** SQL command.

Example

```
sdelindex -names salary_table.primary_key
-database PARIS/hrdb
```

This command deletes the index called *primary_key*, in the table *salary_table*, in the database *hrdb* in the managed SQL Server PARIS.

Permissions

To use **sdelindex**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	index owner

See Also

scheckindex, sgetindex, scrtindex, ssetindex, scopyindex, scompindex

sdellogin

Function

Deletes a SQL Server login from a managed SQL Server.

Syntax

```
sdellogin –names login_names [–server server_name]
   [–version] [–help]
```

Parameters

–names *login_names* – specifies SQL Server logins to delete, where *login_names* is the name of one or more existing logins.

–server *server_name* – specifies the managed SQL Server from which to delete the login, where *server_name* can be any valid managed SQL Server name (see "SQL Server Names" on page 1-15). The default is the current managed SQL Server collection (see "Command Context" on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`sdellogin –name bwebster`**

   Deletes the user login *bwebster* from the current managed SQL Server.

2. **`sdellogin –name bwebster –server KOKO`**

   Deletes the user login *bwebster* from the managed SQL Server KOKO.

Comments

- Before using **sdellogin**, you need the following information:
  - The names of the logins to delete
  - The name of the managed SQL Server from which to delete the logins
- If an error occurs, **sdellogin** rolls back the entire delete operation, restoring all affected objects to their original state.
- You cannot delete a login that:
  - Owns any database objects
  - Is currently connected to SQL Server

  If a login meets either of these conditions when you try to delete it, the **sdellogin** command fails and displays an error message.

Permissions

To use **sdellogin**, you must have the following roles:

TME	ESSM	SQL Server
any	**security**	**sa_role**

See Also

scomplogin, scopylogin, scrtlogin, sgetlogin, smanageserver, ssetlogin

sdelmirror

Function

Deletes a mirror from a device.

Syntax

```
sdelmirror –device device_name {–primary | –secondary}
   [–server server_name] [–version] [–help]
```

Parameters

–device *device_name* – specifies the logical name of the device from which to delete the mirror.

–primary – deletes the primary device. If you delete the primary device, the mirror device becomes the unmirrored, primary device with the same logical device name as *device_name*.

–secondary – deletes the mirror (secondary) device.

–server *server_name* – specifies the managed SQL Server in which *device_name* exists, where *server_name* can be any valid managed SQL Server name (see "SQL Server Names" on page 1-15). The default is the current managed SQL Server collection (see "Command Context" on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
sdeldev –device dev3 –secondary
```

Stops mirroring of the primary device *dev3* in the current managed SQL Server.

Comments

- Before using **sdeldev**, you need the following information:
  - The name of the primary device
  - The name of the managed SQL Server in which the primary device exists
- If an error occurs, **sdelmirror** rolls back the entire delete operation, restoring all affected objects to their original state.

Permissions

To use **sdelmirror**, you must have the following roles:

TME	ESSM	SQL Server
user	space	sa_role

See Also

scrtdev, scompdev, scopydev, sgetdev, ssetdev, scrtmirror

sdelproc

Function

Deletes stored procedures from a database.

Syntax

```
sdelproc -names procedure_names [-database db_path]
   [-version] [-help]
```

Parameters

–names *procedure_names* – specifies the names of the stored procedures to delete. The names of the stored procedures can be prefixed with the name of the owner as follows: owner.procedure_name.

–database *db_path* – specifies the database where the stored procedures reside. The default is the current managed database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
sdelproc -names "my_proc1 old_proc2"
-database BOSTON/test_db
```

This command deletes the procedures *my_proc1* and *old_proc2* from the database *test_db* in the managed SQL Server BOSTON.

Permissions

To use **sdelproc**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	stored procedure owner

See Also

sgetproc, scrtproc, ssetproc, scopyproc, scompproc

sdelrmtlogin

Function

Deletes remote logins from a SQL Server remote access configuration (deletes entries from the *sysremotelogins* table).

Syntax

sdelrmtlogin –names *remote_login_names* [–wildcard]
–remoteserver *remote_server_name*
[–server *server_name*] [–version] [–help]

Parameters

–names *remote_login_names* – specifies the remote logins to delete, where *remote_login_names* lists one or more login names existing in the local SQL Server's *sysremotelogins* table. If the list of remote login names contains spaces, you must enclose the list in double quotation marks (see "Option Lists" on page 1-3).

–wildcard – enables wildcard matching on *remote_login_names* (see "Wildcards" on page 1-4).

–remoteserver *remote_server_name* – specifies the name of the remote SQL Server containing the specified remote logins (*remote_login_names*). The remote SQL Server does not necessarily have to be a managed SQL Server (for more on managed servers, see **smanageserver**).

–server *server_name* – specifies the local SQL Server from which to delete the remote logins, where *server_name* can be any valid managed SQL Server name (see "SQL Server Names" on page 1-15). The default is the current managed SQL Server collection (see "Command Context" on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
sdelrmtlogin –name jimmyh –remoteserver AGRA
–server KOKO
```

Deletes the remote login *jimmyh* from the *sysremotelogins* table of the managed SQL Server KOKO. The login can no longer access the local SQL Server KOKO from the remote SQL Server AGRA.

Comments

- Before using **sdelrmtlogin**, you need the following information:
  - The names of the remote logins to delete
  - The name of the remote SQL Server
  - The name of the local SQL Server from which to delete the remote logins
- If an error occurs, **sdelrmtlogin** rolls back the entire delete operation, restoring all affected objects to their original state.

Permissions

To use **sdelrmtlogin**, you must have the following roles:

TME	ESSM	SQL Server
any	**security**	**sa_role**

See Also

scrtrmtlogin, scrtrmtserver, sdelrmtserver, sgetrmtserver, ssetrmtlogin, ssetrmtserver

sdelrmtserver

Function

Deletes a remote SQL Server entry in the *sysservers* table for a local SQL Server, so that the remote SQL Server may no longer initiate requests in the local SQL Server.

Syntax

```
sdelrmtserver –names remote_server_names
   [–remotelogins] [–server server_name] [–version]
   [–help]
```

Parameters

–names *remote_server_names* – specifies the remote SQL Servers for which the local SQL Server is to deny access, where *remote_server_names* is the name of one or more existing SQL Servers. The remote SQL Server does not necessarily have to be a managed SQL Server (for more on managed servers, see **smanageserver**).

–server *server_name* – specifies the local SQL Server from which to delete the remote SQL Server entry in *sysservers*, where *server_name* can be any valid managed SQL Server name (see “SQL Server Names” on page 1-15). The default is the current managed SQL Server collection (see “Command Context” on page 1-15).

–remotelogins – deletes all remote logins associated with *remote_server_name* from the local SQL Server’s *sysremotelogins* table.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
sdelrmtserver –name AGRA –server KOKO
```

Deletes the remote SQL Server AGRA from the list of servers (*sysservers*) that can access the local SQL Server KOKO.

Comments

- Before using **sdelrmtserver**, you need the following information:
  - The names of the remote SQL Servers to delete
  - The name of the local SQL Server from which to delete the remote SQL Server entries in *sysservers*
- If the local SQL Server has remote logins configured for access from the remote SQL Servers, the **sdelrmtserver** command fails and displays an error message. Either delete the remote logins using the **sdelrmtlogin** command, or specify the **–remotelogins** option for this command.
- If an error occurs, **sdelrmtserver** rolls back the entire delete operation, restoring all affected objects to their original state.

Permissions

To use **sdelrmtserver**, you must have the following roles:

TME	ESSM	SQL Server
any	**security**	**sso_role**

See Also

scrtrmtlogin, scrtrmtserver, sdelrmtlogin, sgetrmtserver, ssetrmtlogin, ssetrmtserver

sdelrule

Function

Deletes rules from a database.

Syntax

```
sdelrule -names rule_names [-database db_path]
   [-version] [-help]
```

Parameters

–names *rule_names* – specifies the names of the rules to delete. The names of the rules can be prefixed with the name of the owner as follows: *owner.rule_name*.

–database *db_path* – specifies the database in which the rules exist. The default is the current managed database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
sdelrule -names "dbo.id_rule sam.sam_rule1"
-data BONN_DEV/results_db
```

This command deletes the rules named ***dbo.id_rule*** **and** ***sam.sam_rule1*** that are in the database *results_db* in the managed SQL Server BONN_DEV. The command unbinds the rules from all existing bindings before deleting them.

Permissions

To use **sdelrule**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	rule owner

See Also

sgetrule, scrtrule, ssetrule, scopyrule, scomprule

sdelseg

Function

Deletes a segment from a database.

Syntax

```
sdelseg –names segment_names [–database db_path]
   [–version] [–help]
```

Parameters

–names *segment_names* – specifies the segments to delete, where *segment_names* are the names of one or more segments to delete from the database's *syssegments* table.

–database *db_path* – specifies the database from which to delete the segment, where *db_path* can be any valid managed database name (see "Database Names" on page 1-16). The default is the current database collection (see "Command Context" on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
sdelseg –name seg1 –database KOKO/pubs2
```

Deletes a segment named *seg1* from the *pubs2* database on the managed SQL Server KOKO.

Comments

- Before using **sdelseg**, you need the following information:
  - The names of the segments to delete
  - The name of the database from which to delete the segments
- If an error occurs, **sdelseg** rolls back the entire delete operation, restoring all affected objects to their original state.
- You cannot delete a segment if it contains database objects. You must first assign the objects to another segment or remove the objects.

  If a segment contains database objects, when you try to delete it, the **sdelseg** command fails and displays an error message.
- If any thresholds exist on the segments being deleted, the **sdelseg** command does not drop the stored procedures associated with those thresholds.

Permissions

To use **sdelseg**, you must have the following roles:

TME	ESSM	SQL Server
any	**space**	**sa_role**

See Also

scompseg, scopyseg, scrtseg, sgetseg, ssetseg

sdelsybaselogin

Function

Removes SQL Server login information for an ESSM administrator.

Syntax

```
sdelsybaselogin –principal login_name
   [[–policyregion region_name]
       | [–server server_name] | –enterprise]
   [–version] [–help]
```

Parameters

–principal *login_name* – specifies the Tivoli name for the user. On UNIX systems, this is the UNIX login name.

–policyregion *region_name* – specifies the specific policy region for which SQL Server login information will be deleted.

–server *server_name* – specifies the managed SQL Server for which SQL Server login information will be deleted.

–enterprise – specifies that only enterprise-level login information be deleted.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
sdelsybaselogin -principal webster -server KOKO
```

Deletes SQL Server login information from the SQL Server KOKO for the ESSM administrator known as *webster* in the TME.

Comment

Omitting the **–policyregion**, **–server**, or **–enterprise** option, deletes all SQL Server login information for the **–principal**.

Permissions

To use **sdelsybaselogin**, you must have the following roles:

TME	ESSM	SQL Server
super	none	none

See Also

scrtlogin, scrtsybaselogin, sdellogin, ssetsybaselogin

sdeltable

Function

Deletes tables from a database.

Syntax

```
sdeltable -names table_names [-database db_path]
   [-version] [-help]
```

Parameters

–names *table_names* – specifies the names of the tables to delete. The names of the tables can be prefixed with the name of the owner as follows: *owner.table_name*.

–database *db_path* – specifies the database in which the tables exist. The default is the current managed database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
sdeltable -names past_employees
-database ARCHIVES/employeeDB
```

This command deletes the table *past_employees* that is located in the database *employeeDB* in the managed SQL Server ARCHIVES.

Permissions

To use **sdeltable**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	table owner

See Also

sgettable, scrttable, ssettable, scopytable, scomptable, schecktable

sdelthresh

Function

Deletes a threshold from a segment in a database.

Syntax

```
sdelthresh –name "segment_name free_pages"
   [–database db_path] [–version] [–help]
```

Parameters

–name – specifies the segment name (*segment_name*) and free page threshold (*free_pages*) of the threshold to delete. You must enclose the *segment_name* and *free_pages* pair in double quotation marks (see “Option Lists” on page 1-3).

segment_name – specifies the name of the segment for which the threshold is defined.

free_pages – specifies the threshold’s limit, in number of pages.

–database *db_path* – specifies the database containing the specified threshold to delete, where *db_path* is a valid managed database name (see “Database Names” on page 1-16). The default is the current database collection (see “Command Context” on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
sdelthresh –name "default 200"
```

Deletes the threshold on the segment *default* (in the current database) that has a page limit of 200 pages.

Comments

- Before using **sdelthresh**, you need the following information:
  - The name of the database containing the segment for which the threshold is defined
  - The name of the segment
  - The page limit at which the threshold is set
- The **sdelthresh** command does not drop the threshold's stored procedure.

Permissions

To use **sdelthresh**, you must have the following roles:

TME	ESSM	SQL Server
any	**space**	**sa_role**

See Also

scompseg, scopyseg, scrtthresh, sgetseg

sdeltrigger

Function

Deletes triggers from a database.

Syntax

```
sdeltrigger -names trigger_names [-database db_path]
   [-version] [-help]
```

Parameters

–names *trigger_names* – specifies the names of the triggers to delete. The names of the triggers can be prefixed with the name of the owner as follows: *owner.trigger_name*.

–database *db_path* – specifies the database in which the triggers exist. The default is the current managed database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
sdeltrigger -names order_delete -database
PARIS/FY95
```

This command deletes the trigger named *order_delete* in the database *FY95* in the managed SQL Server PARIS.

Permissions

To use **sdeltrigger**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	trigger owner

See Also

sgettrigger, scrttrigger, scopytrigger, scomptrigger

sdeluser

Function

Deletes a user from a database so that the associated login no longer has access to the database.

Syntax

```
sdeluser –names user_names [–database db_path]
   [–version] [–help]
```

Parameters

–names *user_names* – specifies the names of the users to delete. After you delete the users, the associated SQL Server logins can no longer access the database.

–database *db_path* – specifies the database from which to delete *user_name,* where *db_path* is a valid managed database name (see "Database Names" on page 1-16). The default is the current database collection (see "Command Context" on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
sdeluser –names harryc –database pubs2
```

Deletes the user *harryc* from the *pubs2* database in the current managed SQL Server. The login that was associated with user *harryc* no longer has access to the *pubs2* database.

Comments

- You cannot delete a user who owns database objects. Either change the ownership (**ssetdb**), delete the objects, or consider locking the user's login (**ssetlogin**).
- Before using **sdeluser**, you need the following information:
  - The names of the database users to delete
  - The name of the database containing the users
- If other logins are aliases of the users being deleted, **sdeluser** deletes the alias mapping (in the system table *sysalternates*) and those logins no longer have access to the database.
- **sdeluser** deletes the alias username along with **all** login relationships to that alias. Use the **ssetuser** command's **–dropaliases** option to remove specific logins from the alias.
- If an error occurs, **sdeluser** rolls back the entire delete operation, restoring all affected objects to their original state.

Permissions

To use **sdeluser**, you must have the following roles:

TME	ESSM	SQL Server
any	**security**	**sa_role** or Database Owner

See Also

scompuser, scopyuser, scrtuser, sgetuser, ssetuser

sdelview

Function

Deletes views from within a database.

Syntax

```
sdelview -names view_names [-database db_path]
   [-version] [-help]
```

Parameters

–names *view_names* – specifies the names of the views to delete. The names of the views can be prefixed with the name of the owner as follows: *owner.view_name*.

–database *db_path* – specifies the database in which the views exist. The default is the current managed database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
sdelview -names top_salaries -database PARIS/hrdb
```

This command deletes the view named *top_salaries* from the database *hrdb* in the managed SQL Server PARIS.

Permissions

To use **sdelview**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	view owner

See Also

sgetview, scrtview, scopyview, scompview

sdumpdb

Function

Backs up a copy of the specified database to a storage device.

Syntax

```
sdumpdb -name db_name [-server server_name]
   [-transaction [-truncate | -backupandlog
       | -logandtruncate]]
   [-density density] [-blocksize blocksize]
   [-capacity capacity] [-dumpvolume dump_volume_name]
   [-file file_name] [-nodismount] [-unload]
   [-retaindays day_count] [-init]
   [-version] [-help]
```

Stripe device information must be specified via standard input and must contain one or more stripe device specifications, as follows:

```
-stripedevice device_name [-backupserver server_name]
   [-dumpvolume volume_name] [-file file_name]
   [-density density] [-blocksize blocksize]
   [-capacity capacity]
```

Parameters

–name *db_name* – specifies the database to restore.

–server *server_name* – specifies the SQL Server installation to which the database will be restored.

–transaction – backs up the transaction log.

–truncate – truncates the transaction log without making a backup copy.

–backupandlog – backs up the transaction log and records the backup in a transaction log entry.

–logandtruncate – backs up the transaction log and removes the inactive portion of the log.

–density *density* – overrides the default density for a tape with a specified density.

–blocksize *blocksize* – overrides the default blocksize for a dump device with a specified blocksize.

–capacity *capacity* – specifies the maximum amount of data that the device can write to a single tape volume.

–dumpvolume *dump_volume_name* – specifies the name of the volume containing the database.

–file *file_name* – specifies the name of the dump file.

–nodismount – specifies that the tape is not dismounted after the database is backed up.

–unload – rewinds the tape when the database is backed up.

–retaindays – specifies the number of days the Backup Server prevents you from overwriting the dump.

–init – re-initializes the tape volume before backing up the database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

–stripedevice *device_name* – specifies the name of the stripe device.

–backupserver *server_name* – specifies the name of the backup SQL Server.

–dumpvolume *volume_name* – specifies the volume.

–file *file_name* – specifies the name of the dump file.

–density *density* – overrides the default density for a tape with a specified density.

–blocksize *blocksize* – overrides the default blocksize for a dump device with a specified blocksize.

–capacity *capacity* – specifies the maximum amount of data that the device can write to a single tape volume.

Example

```
sdumpdb -name pubs2 -server BOSTON <stripe.opt
```

where *stripe.opt* contains the following:

```
-stripedevice sdev1 -backupserver BACKUP
-dumpvolume pvolume
```

Backs up the *pubs2* database to dump device *sdev1* on BACKUP.

Comments

- Before using **sdumpdb** you should know
  - The name of the database to restore
  - The name of the dump device you want to use
  - The size of the database and the capacity of the dump device
- Backup messages are displayed in the Tivoli notification group, TME Administration.

Permissions

To use **sdumpdb**, you must have the following roles:

TME	ESSM	SQL Server
admin, **senior**, or **super**	**dump**	**oper_role**

See Also

scrtdev, scrtmirror, sgetdev, sgetdumpdev, sloaddb, ssetdev, ssetdumpdev, schangevol

sgetcmdperm

Function

Gets information about explicitly granted and revoked permissions on the following commands:

- **create database**
- **create default**
- **create procedure**
- **create rule**
- **create table**
- **create view**
- **dump database**
- **dump transaction**

Syntax

```
sgetcmdperm [-names commands [-wildcard]]
   [-grantees users_or_groups_or_roles_names]
   [-database db_path] [-list]
   [-version] [-help]
```

Parameters

–names *commands* – specifies the command names on which to report.

–wildcard – enables use of wildcard characters in the command name (see “Wildcards” on page 1-4).

–grantees *users_*or*_groups_*or*_roles_names* – specifies the user, group, or role, on which to get permission information (for example, john, consultants, sa_role).

–database *db_path* – specifies the database on which to get permission information. The default is the current database collection (see “Command Context” on page 1-15).

–list – displays output in list format.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
sgetcmdperm -grantee guest -database AGRA/pubs2
```

Reports on all command permissions granted to and revoked from the user *guest* in the *pubs2* database.

```
*** Command Permissions in Database: AGRA/pubs2 ***
Type    Command           Grantee  Grantor
------  ----------------  -------  -------
Grant   Create Default    guest    dbo
Grant   Create Rule       guest    dbo
Grant   Create Table      guest    dbo
Grant   Create View       guest    dbo
Revoke  Create Procedure  guest    dbo
```

Comments

- If you do not supply any command line options, **sgetcmdperm** reports on all command permissions in the default database.
- **sgetcmdperm** returns an empty table or NULL if the specified commands have only implicit permissions granted.

Permissions

To use **sgetcmdperm**, you must have the following roles:

TME	ESSM	SQL Server
any	any	none

See Also

sgetobjperm, ssetcmdperm, ssetobjperm

sgetdatatype

Function

Displays the definition of user datatypes.

Syntax

```
sgetdatatype [-names datatype_names]
   [-onlynames | [ [-summary] [-bindings]
   [-referencedby] [-list] ]]
   [-database db_path] [-wildcard] [-version] [-help]
```

Parameters

–names *datatype_names* – specifies names of the user datatypes. Type names must conform to the rules for identifiers and must be unique for each owner in the database. If you do not specify **–names**, the information is retrieved for all types defined in the specified database.

–onlynames – displays only the names of the datatypes.

–summary – displays summary information for the datatypes. Information includes the datatype name, owner, physical type, length, precision, scale, and allow nulls setting for the datatypes. This is the default option if neither **–bindings** or **–referencedby** is specified.

–bindings – displays the list of objects: defaults, rules, that this data type is bound to. Output is displayed in list mode only. The object name, type, and owner are displayed for each binding.

–referencedby – displays the stored procedures and tables that reference this data type. Output is displayed in list mode only. The object name, type, and owner is displayed for each reference.

–list – displays the output in list format.

–database *db_path* – specifies the database in which the types exist. The default is the current managed database.

–wildcard – enables wildcard pattern matching on the datatype names.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
sgetdatatype -name qa_nvarchar -data BOSTON/pubs2
```

This command displays the definition information for a datatype called *qa_nvarchar* that is located in the database *pubs2,* in the managed SQL Server BOSTON.

```
*** User-Defined Datatypes in Database: JOHNNYP/pubs2 ***
  User Datatype Name Owner  Phystype  Length  Nulls List  Precision  Scale
  ----------------  -----  ------- ------ ---------  --------  -----
  qa_nvarchar       dbo    varchar  255    NoNulls    null      null
```

Permissions

To use **sgetdatatype**, you must have the following roles:

TME	ESSM	SQL Server
any	any	any

See Also

scrtdatatype, sdeldatatype, ssetdatatype, scopydatatype, scompdatatype

sgetdb

Function

Gets information about databases. By default, **sgetdb** displays a summary report of the following attributes for each database:

- Database name
- Database size
- Database Owner
- Creation date
- Number of users
- Dump date

You may also request information about:

- Space utilization
- Database device information
- Database options

Syntax

```
sgetdb [–names db_names]
   [–onlynames | -forload | -suspect]
       | [[-summary] [-list] [-devices] [-space]
           [-config {"%" | names}]]
   [–server server_name] [–version] [–help]
```

Parameters

–names *db_names* – specifies the databases for which you want information, where *db_names* lists one or more existing database names. If the list of database names contains spaces, you must enclose the list in double quotation marks (see “Option Lists” on page 1-3). If you omit this option, **sgetdb** displays information for all databases in SQL Server. Collection path names are invalid here (use **–server** to specify a SQL Server collection object).

–onlynames – displays the names of all databases; the output is in list format.

–forload – displays the names of databases that were created for load.

–suspect – displays the names of databases that are corrupt.

–summary – includes summary report attributes in the output (see "Attribute Summary and Detail Output Options" on page 1-11). If you do not supply a name, **–summary** displays information about databases that are neither corrupted nor for load. If you supply the name of a database that is for load or corrupted, ESSM generates an error message.

–list – displays output in list report format (see "Tabular and List Output Format Options" on page 1-10).

–devices – displays the names of the devices allocated to each database and the size, usage, and available free space for each device.

–space – displays space utilization of each database. The report includes space reserved, space used for data, index size, and unused space.

–config {*"%"* | *names*} – specifies the database options to display for each database, along with their current values, where *names* lists one or more configuration names to display. If the list of database options contains spaces, you must enclose the list in double quotation marks (see "Option Lists" on page 1-3).

The valid database option names are:

abort tran on log full
allow nulls by default
auto identity
dbo use only
ddl in tran
no chkpt on recovery
no free space acctg
read only
select into/bulkcopy
single user
trunc log on chkpt

See Appendix A, "Database Options" for a description of these database options. SQL Server accepts any unique string that is part of the option name. If you specify an option that contains spaces, you must enclose the option in single quotation marks. This option also accepts wildcard characters (%). See "Wildcards" on page 1-4.

–server *server_name* – specifies the managed SQL Server in which the databases reside, where *server_name* can be any valid managed SQL Server name (see "SQL Server Names" on page 1-15). The default is the current managed SQL Server collection (see "Command Context" on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`sgetdb –server KOKO`**

   Displays the summary information for all databases on the managed SQL Server KOKO using tabular output format (by default):

```
*** Database: KOKO/master ***
Size in MB  Owner  Creation Date       Dump Date           Number of Users
----------  -----  ------------------  ------------------  ---------------
         8  sa     Jan  1 1900 12:00AM  Sep 21 1994 11:49AM              7
*** Database: KOKO/model ***
Size in MB  Owner  Creation Date       Dump Date           Number of Users
----------  -----  ------------------  ------------------  ---------------
         2  sa     Jan  1 1900 12:00AM  Sep 21 1994 11:41AM              0
*** Database: KOKO/pubs2 ***
Size in MB  Owner  Creation Date       Dump Date           Number of Users
----------  -----  ------------------  ------------------  ---------------
         2  sa     Sep 21 1994 11:48AM  Sep 21 1994 11:49AM              2
*** Database: KOKO/sales ***
Size in MB  Owner   Creation Date       Dump Date           Number of Users
----------  ------  ------------------  ------------------  ---------------
         2  sybase  Nov 22 1994  2:28PM  Nov 22 1994  2:28PM              0
*** Database: KOKO/sybsecurity ***
Size in MB  Owner  Creation Date       Dump Date           Number of Users
----------  -----  ------------------  ------------------  ---------------
        10  sa     Sep 21 1994 11:41AM  Jan 19 1995  4:28PM              0
*** Database: KOKO/sybsystemprocs ***
Size in MB  Owner  Creation Date       Dump Date           Number of Users
----------  -----  ------------------  ------------------  ---------------
        10  sa     Sep 21 1994 11:36AM  Jan 19 1995  4:28PM              1
*** Database: KOKO/tempdb ***
Size in MB  Owner  Creation Date       Dump Date           Number of Users
----------  -----  ------------------  ------------------  ---------------
         2  sa     Jan 10 1995 11:22AM  Jan 19 1995  4:28PM              1
```

2. **`sgetdb –server KOKO –space`**

   Displays space utilization information for all databases in the managed SQL Server KOKO:

```
*** Database: KOKO/master ***
Reserved in MB Data in MB Index Size in MB Unused in MB
-------------  ---------  ---------------  -----------
      2.06836   0.791016         0.167969      1.10938
*** Database: KOKO/model ***
Reserved in MB Data in MB Index Size in MB Unused in MB
-------------  ---------  ---------------  -----------
     0.652344  0.0585938         0.046875     0.546875
*** Database: KOKO/pubs2 ***
Reserved in MB Data in MB Index Size in MB Unused in MB
-------------  ---------  ---------------  -----------
      1.07227   0.162109        0.0917969     0.818359
*** Database: KOKO/sybsecurity ***
Reserved in MB Data in MB Index Size in MB Unused in MB
-------------  ---------  ---------------  -----------
     0.683594     0.0625         0.046875     0.574219
*** Database: KOKO/sybsystemprocs ***
Reserved in MB Data in MB Index Size in MB Unused in MB
-------------  ---------  ---------------  -----------
      7.78125    7.07227          0.15625     0.552734
*** Database: KOKO/tempdb ***
Reserved in MB Data in MB Index Size in MB Unused in MB
-------------  ---------  ---------------  -----------
     0.699219  0.0917969         0.046875     0.560547
```

3. **sgetdb –name master –devices**

   Displays the devices allocated to provide space for the master database (in the currently managed SQL Server):

```
*** Database Info KOKO/master ***
Device Fragment  Size (MB)  Usage         Free kbytes
---------------  ---------  ------------  -----------
master           3.0        data and log          928
master           5.0        data and log         5120
```

4. **sgetdb –server BURL –onlynames**

   Displays the names of all databases in the managed SQL Server BURL:

```
master
model
pubs2
sybsecurity
sybsystemprocs
tempdb
```

5. **sgetdb –server BOSTON –config % –name pubs2**

   Displays all database configuration options for the *pubs2* database in the managed SQL Server BOSTON:

```
*** Database Info KOKO/pubs2 ***
Name                    Value
---------------------   -----
abort tran on log full      0
allow nulls by default      0
dbo use only                0
ddl in tran                 0
no chkpt on recovery        0
auto identity                0
no free space acctg         0
read only                   0
select into/bulkcopy        0
single user                 0
trunc log on chkpt          0
```

6. **`sgetdb -name pubs2 -config "'ddl in tran'" -server KOKO`**

   Displays the value of the **ddl in tran** option for the *pubs2* database in KOKO.

```
*** Database: KOKO/pubs2 ***
Name         Value
-----------  -----
ddl in tran      0
```

Comments

- If you do not supply any command line options, **sgetdb** reports summary information on all databases in the current managed SQL Server.
- **sgetdb** always evaluates wildcard characters in the **–config** list of database options. Use brackets [ ] to escape evaluation of wildcard characters in the **–config** list (see “Wildcards” on page 1-4).
- The number of database users column in the output indicates the number of users that have the database as their default database.

Permissions

To use **sgetdb**, you must have the following roles:

TME	ESSM	SQL Server
any	any	**sa_role** or valid database user

See Also

scheckdb, scompdb, scopydb, scrtdb, sdeldb, sgetdev, sgetseg, ssetdb

sgetdbaudit

Function

Gets information about command auditing for one or more databases on a managed SQL Server.

Syntax

```
sgetdbaudit -names db_names [-wildcard]
   [-onlynames] [–list] [-server server_name]
   [–version] [–help]
```

Parameters

–names *db_names* – specifies the databases for which you want information, where *db_names* lists one or more existing database names. If the list of database names contains spaces, you must enclose the list in double quotation marks (see “Option Lists” on page 1-3). If you omit this option, **sgetdbaudit** displays information for all databases in SQL Server. Collection path names are invalid here (use **–server** to specify a SQL Server collection object).

–wildcard – enables wildcard matching on *db_names* (see “Wildcards” on page 1-4).

–onlynames – displays only the names of the audited databases; the output is in list format.

–list – displays the report in list format (see “Tabular and List Output Format Options” on page 1-10).

–server *server_name* – specifies the managed SQL Server in which the audited databases reside, where *server_name* can be any valid managed SQL Server name (see “SQL Server Names” on page 1-15). The default is the current managed SQL Server collection (see “Command Context” on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

`sgetdbaudit -names pubs2`

Displays the current command auditing configuration for the *pubs2* database on the current SQL Server:

```
*** Database Auditing in Server: AGRA***
Name   Drop  Use  Access  Grant  Revoke  Truncate Table
-----  ----  ---  ------  -----  ------  --------------
pubs2   Off  Off     Off    Off     Off             Off
```

Comment

Before using **sgetdbaudit**, you need the names of the databases from which to get auditing information.

Permissions

To use **sgetdbaudit**, you must have the following roles:

TME	ESSM	SQL Server
any	any	**sso_role**

See Also

scrtauditrec, sgetloginaudit, sgetprocaudit, sgetroleaudit, sgetserveraudit, sgettableaudit, sinstallaudit, ssetdbaudit, ssetloginaudit, ssetprocaudit, ssetroleaudit, ssetserveraudit, ssettableaudit

sgetdefault

Function

Displays the defaults in a database.

Syntax

```
sgetdefault [-names default_names]
   [-onlynames | [-summary] [-bindings] [-list] ]
   [-database db_path] [-wildcard] [-version] [-help]
```

Parameters

–names *default_names* – specifies the names of the defaults to retrieve information on. The name of the defaults can be prefixed with the name of the owner as follows: owner.default_name. If you do not specify **–names**, the information is retrieved for all defaults defined in the specified database.

–onlynames – displays only the names of the defaults.

–summary – displays summary information for the defaults. Information includes the default name, owner, creation date and value for the defaults. This is the default format.

–bindings – displays the list of objects: user datatypes and table columns that this default is bound to. Output is displayed in list mode only. The bound object's name, type, and owner are displayed for each binding.

–list – displays the output in list format.

–database *db_path* – specifies the database in which the defaults exist. The default is the current managed database.

–wildcard – enables wildcard pattern matching on the default names.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

`sgetdefault -names today -database PARIS/master`

This command displays the default *today* in the database *master,* in the managed SQL Server PARIS.

```
*** Defaults in Database: PARIS/master ***
Default Name: today
Owner Name: dbo
Default Create Date: Oct 23 1995  1:40PM
Text summary:
    Default text: create default today as getdate()
```

Permissions

To use **sgetdefault**, you must have the following roles:

TME	ESSM	SQL Server
any	any	any

See Also

scrtdefault, sdeldefault, ssetdefault, scopydefault, scompdefault

sgetdev

Function

Gets information about devices in a managed SQL Server. By default, **sgetdev** displays a summary report of the following device attributes:

- Logical device name
- Physical device name
- Size
- Default pool
- Disk controller
- Starting virtual address
- Virtual device number
- Reserved space in MB
- Unused space in MB

You can also request information about:

- Database information
- Space allocation
- Mirror configuration

Syntax

```
sgetdev [–names device_names [–wildcard]]
   [–onlynames | [–database] [–alloc] [–mirror]
       [–summary]]
   [–list] [–server server_name] [–version] [–help]
```

Parameters

–names *device_names* – specifies the database devices for which you want information, where *device_names* lists one or more database device names. Without this option, **sgetdev** reports on all database devices occurring on SQL Server. If the list of device names contains spaces, you must enclose the list in double quotation marks (see “Option Lists” on page 1-3).

–wildcard – enables wildcard matching on *device_names* (see “Wildcards” on page 1-4).

–onlynames – displays only the names of the logical devices; the output is in list format.

–summary – includes summary report attributes in the output (see “Attribute Summary and Detail Output Options” on page 1-11).

–database – displays database information for each device, including the database names on the device, the storage type for each database, and space allocated to each database.

–alloc – displays space allocation information for each device, including space currently allocated and page range.

–mirror – displays mirror configuration information for each device, including physical name, I/O type, and status.

–list – displays output in list report format (see “Tabular and List Output Format Options” on page 1-10).

–server *server_name* – specifies the managed SQL Server in which the devices reside, where *server_name* can be any valid managed SQL Server name (see “SQL Server Names” on page 1-15). The default is the current managed SQL Server collection (see “Command Context” on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`sgetdev`**

   Displays summary information for all devices in the current managed SQL Server:

```
*** Database Devices in KOKO ***
Device       Physical Name                        Size in MB  VDevNo
Status   Vstart  Kind       Reserved in MB  Unused in MB
-----------  ----------------------------------  ----------  ------  -
------  ------  ---------  -------------  ------------
data_dev1    /work2/system1001/KOKO_data_dev1.dat          20       3
default       0  DB Device            8         5.516
log_dev1     /work2/system1001/KOKO_log_dev1.dat           10       4
default       0  DB Device            0            0
master       d_master                                      17
0                0  DB Device            16.5           13
sybsecurity  /work2/system1001/KOKO_AUDIT.dat              10
2                0  DB Device               0            0
sysprocsdev  /work2/system1001/KOKO_PROCS.dat              10
1                0  DB Device              10         2.172
test_dev1    /work2/system1001/KOKO_test_dev1.dat           2
5                0  DB Device               2            2
test_dev2    /work2/system1001/KOKO_test_dev2.dat           2
6                0  DB Device               0            0
```

2. **sgetdev –onlynames –server KOKO**

   Displays the names of all devices in the managed SQL Server KOKO:

   ```
   data_dev1
   log_dev1
   master
   sybsecurity
   sysprocsdev
   test_dev1
   test_dev2
   ```

3. **`sgetdev -alloc -name test% -wildcard -server KOKO`**

   Displays space allocation information for all device names beginning with "test" in the managed SQL Server KOKO:

```
*** Database Devices in KOKO ***

Device Name: test_dev1
Physical Name: /work2/system1001/KOKO_test_dev1.dat
Size in Megabytes: 2
Virtual Device Number: 5
Status:
Virtual Start: 0
Controller Type: DB Device
Reserved Space in Megabytes: 2
Unused Space in Megabytes: 2
Fragments:
    Logical Start: 83886080
    Database Name: pubs2
    Size in Megabytes: 2
    Unused Space in Megabytes: 2
    Segment Map: 3
    Database Id: 6
Device Name: test_dev2
Physical Name: /work2/system1001/KOKO_test_dev2.dat
Size in Megabytes: 2
Virtual Device Number: 6
Status:
Virtual Start: 0
Controller Type: DB Device
Reserved Space in Megabytes: 0
Unused Space in Megabytes: 0
Fragments:
    Logical Start: 100663296
    Database Name: null
    Size in Megabytes: 2
    Unused Space in Megabytes: 2
    Segment Map: 0
    Database Id: null
```

Comment

If you do not supply any command line options, **sgetdev** reports on all devices in the current managed SQL Server.

Permissions

To use **sgetdev**, you must have the following roles:

TME	ESSM	SQL Server
any	any	none

See Also

scompdev, scopydev, scrtdev, scrtmirror, sdeldev, ssetdev

sgetdumpdev

Function

Gets information about dump devices on a managed SQL Server. By default, **sgetdumpdev** displays a summary report of the following device attributes:

- Logical device name
- Physical device name
- Type of device (Tape or disk)
- Size in MB (null for disks)

Syntax

```
sgetdumpdev [-names device_names [-wildcard]]
   [-onlynames] [-list] [-server server_name]
   [-version] [-help]
```

Parameters

–names *device_names* – specifies the dump devices for which you want information, where *device_names* lists one or more dump device names. Without this option, **sgetdumpdev** compares all dump devices occurring on either SQL Server. If the list of device names contains spaces, you must enclose the list in double quotation marks.

–wildcard – enables wildcard matching on *device_names* (see "Wildcards" on page 1-4).

–onlynames – displays only the names of the dump devices. The output is in list format.

–list – displays output in list format.

–server *server_name* – specifies the managed SQL Server in which the devices reside, where *server_name* can be any valid managed SQL Server name (see "SQL Server Names" on page 1-15). The default is the current managed SQL Server collection (see "Command Context" on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
sgetdumpdev -names tapedump1 -server AGRA
```

Gets information about the dump device *tapedump1* on the managed SQL Server AGRA.

```
*** Dump Devices in Server: AGRA ***
Device     Physical Name  Kind  Size in MB
---------  -------------  ----  ----------
tapedump1  /dev/rmt4      Tape         625
```

Comment

If you do not supply any arguments, **sgetdumpdev** reports on all the dump devices in the current managed SQL Server.

Permissions

To use **sgetdumpdev**, you must have the following roles:

TME	ESSM	SQL Server
any	any	none

See Also

sgetdev, scompdumpdev, scopydumpdev, scrtdumpdev, sdeldumpdev

sgetgroup

Function

Gets information about groups in a database. By default, **sgetgroup** displays the current user members for each group.

Syntax

```
sgetgroup [–names group_names [–wildcard]]
   [–onlynames] [–list] [–database db_path]
   [–version] [–help]
```

Parameters

–names *group_names* – specifies the group names for which you want information, where *group_names* lists one or more existing group names. Without this option, **sgetgroup** reports on all groups occurring in the database. If the list of group names contains spaces, you must enclose the list in double quotation marks (see "Option Lists" on page 1-3).

–wildcard – enables wildcard matching on *group_names* (see "Wildcards" on page 1-4).

–onlynames – displays only the names of the groups; the output is in list format.

–list – displays output in list report format (see "Tabular and List Output Format Options" on page 1-10).

–database *db_path* – specifies the database in which the specified groups reside, where *db_path* can be any valid managed database name (see "Database Names" on page 1-16). The default is the current database collection (see "Command Context" on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`sgetgroup –database KOKO/master`**

   Gets information on all groups in the *master* database in the managed SQL Server KOKO.

```
*** Groups In Database: KOKO/master ***
Group Name: altos
Users:
    User: jhodges
    User: ohardwick
Group Name: public
Users:
    User: barneyb
    User: dbo
    User: guest
    User: harryc
    User: probe
Group Name: tenors
Users:
    User: bwebster
```

2. **sgetgroup –names altos**

   Gets information on the group *altos* in the current managed database:

```
*** Groups In Database: KOKO/master ***
Group Name: altos
Users:
    User: jhodges
    User: ohardwick
```

Comment

If you do not supply any command line options, **sgetgroup** reports on all groups in the current managed database.

Permissions

To use **sgetgroup**, you must have the following roles:

TME	ESSM	SQL Server
any	any	**sso_role** or valid database user

See Also

scompgroup, scopygroup, scrtgroup, sgetgroup, sgetuser, ssetgroup

sgetindex

Function

Displays information about the indexes on a database table.

Syntax

```
sgetindex [-names index_names]
   [-onlynames | [-summary] [-columns] [-segments]
       [-list]]
   [-database db_path] [-wildcard] [-version] [-help]
```

Parameters

–names *index_names* – specifies the names of the indexes to retrieve information on in the format owner.table.indexname. If you do not use this parameter, the information is retrieved for all indexes defined in the default or specified database.

Any of the three parts of *index_names* can be replaced with a wildcard. The owner and table name are optional.

–onlynames – displays only the names of the indexes.

–summary – Displays summary information for the indexes. This command shows the index name, table the index is on, index type, unique setting, duplicate key setting, and duplicate rows setting for the indexes. This is the default option.

–columns – Displays the columns that comprise the index.

–segments – Displays the segments on which the indexes reside.

–list – displays the output in list format.

–database *db_path* – specifies the database in which the indexes exist. The default is the current managed database.

–wildcard – enables wildcard pattern matching on the index names.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
sgetindex -data BOSTON/pubs2
-name sales.salesind -segment -columns
```

This command displays information about the index *salesind* in the table *sales* in the database *pubs2* in the managed SQL Server BOSTON.

```
*** Indexes in JOHNNYP_DEV/pubs2_test ***
Index Name: salesind
Owner Name: dbo
Table Name: sales
Unique: True
Clustered: True
Ignore Duplicate Keys: False
Duplicate Rows: Not Allowed
Index Columns:
    Column Name: ord_num
    Column Name: stor_id
Index Segments:
    Segment Name: dataseg_2
```

Comment

You can omit the first two parts (owner and table name) of the index_names argument. The name "a.b" implies that the owner was omitted. The name "a" implies that both the owner and table were omitted. If you specify just the index name, all indexes named with that name are retrieved regardless of their owner or the table they are defined on.

Permissions

To use **sgetindex**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	any

See Also

scheckindex, scrtindex, sdelindex, ssetindex, scopyindex, scompindex

sgetlogin

Function

Gets information about SQL Server login accounts. By default, **sgetlogin** displays a summary report of the following attributes for each login:

- Login name
- Full name
- Default database
- Default language
- Lock status
- Connection information
- Server user ID

You may also request information about:

- Roles
- Database ownership
- Password status information

Syntax

```
sgetlogin [–names login_names [–wildcard]]
   [–onlynames | [-roles] [–ownershipinfo]
       [-passwordinfo] [–summary] [–list]]
   [–server server_name] [–version] [–help]
```

Parameters

–names *login_names* – specifies the logins for which you want information, where *login_names* lists one or more existing SQL Server login names. If the list of login names contains spaces, you must enclose the list in double quotation marks (see “Option Lists” on page 1-3). If you omit this option, **sgetlogin** displays information for all logins in SQL Server.

–wildcard – enables wildcard matching on *login_names* (see “Wildcards” on page 1-4).

–onlynames – displays only the names of the logins; the output is in list format.

–roles – displays SQL Server role assignments for each login.

–ownershipinfo – displays database ownership information for each login.

–passwordinfo – displays password expiration for each login.

–summary – includes summary report attributes in the output (see “Attribute Summary and Detail Output Options” on page 1-11).

–list – displays output in list report format (see “Tabular and List Output Format Options” on page 1-10).

–server *server_name* – specifies the managed SQL Server in which the logins reside, where *server_name* can be any valid managed SQL Server name (see “SQL Server Names” on page 1-15). The default is the current managed SQL Server collection (see “Command Context” on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`sgetlogin`**

   Displays summary information about the logins in the current managed SQL Server:

```
*** Logins in Server: AGRA ***
Login     Full Name       Default Database  Language  Lock
Connected  Server User ID
--------  --------------  ----------------  --------  --------
---------  --------------
harryc    Harry Carney    pubs2             null      unlocked
no                     4
jhodges   Johnny Hodges   pubs2             null      unlocked
no                     5
jimmyh    Jimmy Hamilton  pubs2             null      unlocked
no                     6
```

```
paulgonz  Paul Gonzalves   pubs2            null      unlocked
no                       7
probe     null             master           null      unlocked
no                       2
russellp  Russell Procope  pubs2            null      unlocked
no                       8
sa        null             master           null      unlocked
no                       1
sybase    null             master           null      unlocked
yes                      3
```

2. **sgetlogin –name harryc –summary –ownershipinfo –passwordinfo**

   Displays all available attribute information (summary, ownership, and password) about the login *harryc* in the current managed SQL Server:

   ```
   *** Logins in Server: AGRA ***
   Login Name: harryc
   Login Full Name: Harry Carney
   Login Default Database: pubs2
   Default Language: null
   Login Lock: unlocked
   Password Date: 09/21/94 11:49:17
   Password Expiration Date: null
   Connected: no
   Server User ID: 4
   Databases owned:
       Database: engagements
       Database: instruments
   ```

3. **sgetlogin –onlynames –server DUKE**

   Displays the names of all logins in the managed SQL Server DUKE:

   ```
   *** Logins on Server: DUKE ***
   hcarney
   jhodges
   jimmyh
   paulg
   russellp
   ```

Comments

- If you do not supply any command line options, **sgetlogin** reports on all logins in the current managed SQL Server.
- The **sgetlogin** command does not display passwords.
- The **sgetlogin** command automatically includes summary information when you select the **–ownershipinfo** or **–passwordinfo**

options.
The **–roles** options returns a report that includes the roles, login name and server user ID (suid) information.

Permissions

To use **sgetlogin**, you must have the following roles:

TME	ESSM	SQL Server
any	any	**sso_role.** To use the **–roles** option, you must also have **sa_role**

See Also

scomplogin, scopylogin, scrtlogin, sdellogin, sgetloginaudit, sgetuser, ssetlogin

sgetloginaudit

Function

Gets information about command auditing for one or more logins on a managed SQL Server. By default, **sgetloginaudit** displays a summary report of the following auditing options:

- Table auditing
- View auditing
- Batch auditing (command text)

Syntax

```
sgetloginaudit [–names login_names [–wildcard]]
   [–onlynames | -list] [–server server_name]
   [–version] [–help]
```

Parameters

–names *login_names* – specifies the logins to get auditing information about, where *login_names* lists one or more existing SQL Server login names. If the list of login names contains spaces, you must enclose the list in double quotation marks (see “Option Lists” on page 1-3). If you omit this option, **sgetloginaudit** displays information for all logins in SQL Server.

–wildcard – enables wildcard matching on *login_names* (see “Wildcards” on page 1-4).

–onlynames – displays only the names of the audited logins; the output is in list format.

–list – displays output in list report format (see “Tabular and List Output Format Options” on page 1-10).

–server *server_name* – specifies the managed SQL Server in which the audited logins reside, where *server_name* can be any valid managed SQL Server name (see “SQL Server Names” on page 1-15). The default is the current managed SQL Server collection (see “Command Context” on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`sgetloginaudit`**

   Displays the auditing configuration for all logins in the current SQL Server:

```
*** Login Auditing in Server: KOKO ***
Name        Table  View  Batch
----------  -----  ----  -----
harryc        Off   Off     On
jhodges       Off   Off    Off
jimmyh        Off    On    Off
paulg         Off    On    Off
russellp      Off   Off    Off
```

2. **`sgetloginaudit –names billys –server UMMG`**

   Displays the auditing configuration for login *billys* in the managed SQL Server UMMG:

```
*** Login Auditing in Server: UMMG ***
Name        Table  View  Batch
----------  -----  ----  -----
billys        Off   Off    Off
```

Comments

- If you do not supply any command line options, **sgetloginaudit** reports on all logins in the current managed SQL Server.
- See **sgetroleaudit** and **ssetroleaudit** to get and set auditing of privileged command use for any roles (**sa_role**, **sso_role**, **oper_role**) that may be assigned to a login.

Permissions

To use **sgetloginaudit**, you must have the following roles:

TME	ESSM	SQL Server
any	any	**sso_role** or **sa_role**

See Also

scrtauditrec, sgetdbaudit, sgetprocaudit, sgetroleaudit, sgetserveraudit, sgettableaudit, sinstallaudit, ssetdbaudit, ssetloginaudit, ssetprocaudit, ssetroleaudit, ssetserveraudit, ssettableaudit

sgetobjperm

Function

Gets information about explicitly granted permissions on the following objects:

- Tables
- Views
- Procedures

Syntax

```
sgetobjperm [-names objects [-wildcard]]
  [-grantees users_or_groups_or_roles_names]
  [-permissions] [-nopermissions]
  [-database db_path] [-list]
  [-version] [-help]
```

Parameters

–names *objects* – specifies the names of the objects. If no objects are specified, object permissions are returned for all objects.

–wildcard – enables use of wildcard characters in the *objects* (see "Wildcards" on page 1-4).

–grantees *users_or_groups_or_roles_names* – specifies the user, group, or role, on which to get object permission information (for example, *john*, *consultants*, or *sa_role*). By default, **sgetobjperm** displays permission information for all users, groups, and roles.

–permission – returns all explicitly granted and revoked permissions.

–nopermission – returns objects for which no permissions are explicitly set.

–database *db_path* – specifies the database from which to get permission information, where *db_path* can be any valid managed database name (see "Database Names" on page 1-16). The default is the current database collection (see "Command Context" on page 1-15).

–list – displays output in list format.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`sgetobjperm -names sp_configure -grantees harryc -database AGRA/master`**

   Returns information on permission granted to *harryc* on the *sp_configure* procedure in the *master* database.

```
*** Object Permissions in Database: AGRA/master ***
Object        Type  Action  Grantee  Grantable  Grantor  Permission
------------  ----  ------  -------  ---------  -------  ----------
sp_configure  P     null    null     null       null     null
```

   If *harryc* had **execute** permission on the procedure, “Execute” would appear in the “Action” column and the output would look something like this:

```
*** Object Permissions in Database: AGRA/master ***
Object        Type  Action  Grantee  Grantable  Grantor  Permission
------------  ----  ------  -------  ---------  -------  ----------
sp_configure  P      Execute harryc  NO         dbo      Grant
```

2. **`sgetobjperm -nopermissions -database AGRA/pubs2 -list`**

   Returns a list of objects on which no permission has been granted.

```
*** Object Permissions in Database: AGRA/master***
Object Type: titleview
Permission Type: V
Action Type: null
Grantee Name: null
Grantable: null
Grantor Name: null
Permission Type: null

Object Type: totalsales_trig
Permission Type: TR
Action Type: null
Grantee Name: null
Grantable: null
```

```
Grantor Name: null
Permission Type: null

Object Type: typedflt
Permission Type: D
Action Type: null
Grantee Name: null
Grantable: null
Grantor Name: null
Permission Type: null
```

Comments

- If you do not supply any command line options, **sgetobjperm** reports on all object permissions in the current database.
- **sgetobjperm** returns an empty table or NULL if the specified objects have only implicit permissions granted.

Permissions

To use **sgetobjperm**, you must have the following roles:

TME	ESSM	SQL Server
any	any	**sso_role** or **sa_role**

See Also

sgetcmdperm, ssetcmdperm, ssetobjperm

sgetprf

Function

Gets information about the names contained in a specific ESSM profile of a specified type.

Syntax

```
sgetprf -name profile_name -type profile_type
   [-previewdist [subscribers]] [-version] [-help]
```

Parameters

–name *profile_name* – specifies the name of the profile.

–type *profile_type* – specifies the type of the profile. There are nine ESSM profile types. The SQL Server profile types are:

- SQLServerProfile
- SQLLoginProfile
- SQLDumpDevicesProfile
- SQLDbDevicesProfile
- SQLRemoteServerProfile
- SQLDatabaseProfile
- SQLCacheProfile

The database profile types are:

- SQLUserProfile
- SQLGroupProfile
- SQLSegmentProfile
- SQLDatabaseProfile
- SQLDefaultProfile
- SQLRuleProfile
- SQLDataTypeProfile
- SQLTableProfile
- SQLIndexProfile
- SQLViewProfile
- SQLProcedureProfile

- SQLTriggerProfile

-previewdist [*subscribers***]** – displays the actions that will occur if the specified profile is distributed. The **subscribers** value is a list of subscribers to preview the distribution for. The list can include potential subscribers as well as current subscribers. If the **subscribers** argument is "", then the distribution is previewed for all subscribers to that profile's profile manager.

➤ ***Note***

The output of this option does not tell you whether or not the distribution will succeed. It shows what ESSM will try to do if this profile is distributed.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`sgetprf -name Groups -type SQLGroupProfile`**

   Displays the group names contained in an ESSM group profile called *Groups*.

   ```
   *** Profile Groups ***

   public
   sales
   ```

2. **`sgetprf -name login_profile -type SQLLoginProfile -previewdist Subscriber: BONN_DEV Profile: <login_profile>`**

   This example demonstrates the output of the **sgetprf** command when you request a preview of a distribution.

   ```
   To be CREATED on subscriber BONN_DEV:

      add_login
      xxxxx1
      xxxxx2
      xxxxx3

   To be DELETED from subscriber BONN_DEV:

      delete_login
      yyyyy1
      yyyyy2
      yyyyy3
   ```

```
To be MODIFIED on subscriber BONN_DEV:
   modify_login
   unchanged_login
   zzzzz1
   zzzzz2
   zzzzz3
***End of distibution preview*********************
```

Comments

- The names denote an element in the managed SQL Server or database associated with the profile and are used as keys to retrieve information from the Server or database and distribute it to subscribers.
- The same name cannot be in two different profiles in the same profile manager.

Permissions

To use **sgetprf**, you must have the following roles:

TME	ESSM	SQL Server
any	none	none

See Also

scrtprfmgr, spopulateprf

sgetproc

Function

Displays the definition of stored procedures.

Syntax

```
sgetproc [-names procedure_names]
   [-onlynames | [[-summary] [-bindings]
       [-referencedby]]]
   [-database db_path] [-wildcard] [-version] [-help]
```

Parameters

–names *procedure_names* – specifies the names of the stored procedures to retrieve information about. The name of the stored procedures can be prefixed with the name of the owner as follows: owner.procedure_name. If you do not specify **–names**, the information is retrieved for all stored procedures defined in the specified database.

–onlynames – displays only the names of the stored procedures.

–summary – displays summary information for the stored procedures. Information includes the stored procedure name, owner, date created, procedure SQL and parameter definitions for the stored procedures. The parameter definitions include the parameter name, type, length, precision, scale, default value, and output status for each parameter. This is the default parameter if neither **–bindings** nor **–referencedby** is specified.

–bindings – displays the list of objects: tables, views, datatypes, and stored procedures that this stored procedure refers to. The object name, type, and owner are displayed for each binding.

–referencedby – displays the stored procedures and triggers that reference this stored procedure. The object name, type, and owner are displayed for each reference.

–database *db_path* – specifies the database in which the stored procedures reside. The default is the current managed database.

–wildcard – enables wildcard pattern matching on the stored procedure names.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

➤ ***Note***

This command displays in list mode only. There is no tabular output.

Example

`sgetproc -name dbo.sp_view -database BOSTON/pubs2`

This command displays the definition of the procedure *sp_view* in the database *pubs2* in the managed SQL Server BOSTON.

```
*** Stored Procedures in Database: JOHNNYP/pubs2 ***
Procedure Name: sp_view
Owner Name: dbo
Procedure Create Date: Jul  3 1995  3:20PM
Summary Text:
/* Create a stored procedure that selects from a view */
create proc sp_view
as
      select * from view_authors
      return
```

Permissions

To use **sgetproc**, you must have the following roles:

TME	ESSM	SQL Server
any	any	any

See Also

scrtproc, sdelproc, scopyproc, scompproc

sgetprocaudit

Function

Gets information about the auditing status (on or off) of stored procedures and triggers in a database.

Syntax

```
sgetprocaudit [–names proc_names [–wildcard]]
   [–new] [–onlynames] [–database db_path] [-list]
   [–version] [–help]
```

Parameters

–names *proc_names* – specifies the stored procedures and triggers on which to get auditing information, where *proc_names* lists one or more existing stored procedure or trigger names. If the list contains spaces, you must enclose it in double quotation marks (see "Option Lists" on page 1-3). If you omit this option, **sgetprocaudit** displays auditing information for all stored procedures and triggers in SQL Server.

–wildcard – enables wildcard matching on *proc_names* (see "Wildcards" on page 1-4).

–new – displays the current default auditing settings for newly created stored procedures and triggers. By default, all new stored procedures and triggers adopt these default auditing settings.

–onlynames – displays only the names of audited stored procedures and triggers; the output is in list format.

–database *db_path* – specifies the database in which the audited stored procedures and triggers reside, where *db_path* can be any valid managed database name (see "Database Names" on page 1-16). The default is the current database collection (see "Command Context" on page 1-15).

–list – displays output in list report format (see "Tabular and List Output Format Options" on page 1-10).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
sgetprocaudit -database KOKO/master
```

Displays the current auditing configuration for all procedures and triggers in *master* database on managed SQL Server KOKO:

```
*** Procedure/Trigger Auditing in Database:
KOKO/master ***
Name              Execution
----------------  ---------
sp_configure      Off
sp_dboption       Off
sp_getmessage     Off
sp_procxmode      Off
sp_prtsybsysmsgs  Off
sp_validlang      Off
```

Comment

If you do not supply any command line options, **sgetprocaudit** reports on all stored procedures and triggers in the current managed database.

Permissions

To use **sgetprocaudit**, you must have the following roles:

TME	ESSM	SQL Server
any	any	**sso_role**

See Also

scrtauditrec, sgetdbaudit, sgetloginaudit, sgetroleaudit, sgetserveraudit, sgettableaudit, sinstallaudit, ssetdbaudit, ssetloginaudit, ssetprocaudit, ssetroleaudit, ssetserveraudit, ssettableaudit

sgetprocess

Function

Gets information about SQL Server processes. By default, **sgetprocess** displays a summary report of the following process information:

- Process ID
- Running status
- Login
- Program name
- Command associated with each process

You may also request information about:

- Resource utilization information (physical I/O, CPU time, and memory usage)
- Locks held on each process
- Blocking status
- Execution status

Syntax

```
sgetprocess [–ids process_ids] [–summary]
   [–resources] [–locks] [–block] [–exec] [–list]
   [–server server_name] [–version] [–help]
```

Parameters

–ids *process_ids* – specifies the SQL Server process IDs for which you want information, where *process_ids* lists one or more existing process ID numbers. If the list of ID numbers contains spaces, you must enclose the list in double quotation marks (see “Option Lists” on page 1-3). If you omit this option, **sgetprocess** displays information for all processes in SQL Server.

–summary – includes summary report items in the output (see “Attribute Summary and Detail Output Options” on page 1-11).

–resources – displays process resource utilization information such as physical I/O count (number of reads and writes), CPU time (in milliseconds), memory usage (in pages), and packet size.

–locks – displays information about locks held on each process, including the type of lock, database, table, page, and class associated with each lock.

–block – displays information about processes that are blocking other processes.

–exec – displays information about program execution for each process.

–list – displays output in list report format (see “Tabular and List Output Format Options” on page 1-10).

–server *server_name* – specifies the managed SQL Server in which the processes reside, where *server_name* can be any valid managed SQL Server name (see “SQL Server Names” on page 1-15). The default is the current managed SQL Server collection (see “Command Context” on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`sgetprocess`**

   Displays a summary report of all processes in the current managed SQL Server:

```
*** Processes in Server UMMG ***
PID  Process Status  Login  Program Name  Command
---  --------------  -----  ------------  ----------------
  1  running         billys               SELECT
  2  sleeping        null                 NETWORK HANDLER
  3  sleeping        null                 MIRROR HANDLER
  4  sleeping        null                 AUDIT PROCESS
  5  sleeping        null                 CHECKPOINT SLEEP
```

2. **`sgetprocess -block -server UMMG`**

   Displays details about each the blocking status of each process in the managed SQL Server UMMG:

```
*** Processes in Server UMMG ***
PID  Blocking Process  Time Blocked
---  ----------------  ------------
  1                 0          null
  2                 0          null
  3                 0          null
  4                 0          null
  5                 0          null
```

3. **`sgetprocess -ids 1 -server UMMG -list`**

   Displays a summary report of process ID 1 in the SQL Server UMMG. The report is in list format:

```
*** Processes in Server UMMG***
Process ID: 1
Process ID Status: running
Login Name: sybase
Name of Front End Module:
Command Currently Executed: SELECT
```

Comments

- If you do not supply any command line options, **sgetprocess** reports on all databases in the current managed SQL Server.
- Process status can be one of the following values:

```
infected
background
recv sleep
send sleep
alarm sleep
lock sleep
sleeping
runnable
running
stopped
bad status
log suspend
```

Permissions

To use **sgetprocess**, you must have the following roles:

TME	ESSM	SQL Server
any	any	none

See Also

sgetserver, skillprocess, sstartserver, sstopserver

sgetrmtserver

Function

Gets information about remote servers configured for a specified local SQL Server. By default, **sgetrmtserver** displays a summary report of the following remote SQL Server attributes:

- Remote SQL Server name
- Network password encryption
- Timeout
- Default login map
- Default login

Optionally, the report can include information on the remote servers' remote logins.

Syntax

```
sgetrmtserver [–names remote_server_names [–wildcard]]
   [–onlynames | [–remotelogins] [–summary] [–list]]
   [–server server_name] [–version] [–help]
```

Parameters

–names *remote_server_names* – specifies the remote servers for which you want information, where *remote_server_names* lists one or more remote SQL Server names. If the list of remote servers contains spaces, you must enclose the list in double quotation marks (see "Option Lists" on page 1-3). If you omit this option, **sgetrmtserver** displays information for all remote servers listed in the local SQL Server.

–wildcard – enables wildcard matching on *remote_server_names* (see "Wildcards" on page 1-4).

–onlynames – displays only the names of the remote servers; the output is in list format.

–remotelogins – displays the remote logins associated with each remote SQL Server.

–summary – includes summary report attributes in the output (see "Attribute Summary and Detail Output Options" on page 1-11).

–list – displays output in list report format (see “Tabular and List Output Format Options” on page 1-10).

–server *server_name* – specifies the managed SQL Server in which the remote servers are listed, where *server_name* can be any valid managed SQL Server name (see “SQL Server Names” on page 1-15). The default is the current managed SQL Server collection (see “Command Context” on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **sgetrmtserver**

   Displays summary information in tabular format about the remote servers configured in the current managed SQL Server:

```
*** Remote Servers in Server: UMMG ***
Remote Server  Encrypt  Timeout  Default Login Map  Default Login
-------------  -------  -------  -----------------  -------------
SYB_BACKUP       False    True                None  null
```

2. **sgetrmtserver –server KOKO –list**

   Displays a summary report of each remote SQL Server with access to the managed SQL Server KOKO. The display is in list format:

```
*** Remote Servers in Server: KOKO ***

Remote Server Name: AGRA
Encrypt Passwords: False
Connection Timeout: True
Default Remote Login Mapping: None
Default Local Login Name: null

Remote Server Name: SYB_BACKUP
Encrypt Passwords: False
Connection Timeout: True
Default Remote Login Mapping: None
Default Local Login Name: null
```

3. **sgetrmtserver –server UMMG –onlynames**

   Displays the name of each SQL Server with remote access to the managed SQL Server UMMG:

```
SYB_BACKUP
```

Comments

- If you do not supply any command line options, **sgetrmtserver** reports on all databases in the current managed SQL Server.
- To get information on remote access options for the local SQL Server (such as remote access status and maximum remote logins, servers, and connections), use the **sgetserver** command.

Permissions

To use **sgetrmtserver**, you must have the following roles:

TME	ESSM	SQL Server
any	any	none

See Also

scrtrmtlogin, scrtrmtserver, sdelrmtlogin, sdelrmtserver, ssetrmtlogin, ssetrmtserver

sgetroleaudit

Function

Gets information about status of privileged command auditing for SQL Server roles (**sa_role**, **sso_role**, **oper_role**) in a managed SQL Server.

Syntax

```
sgetroleaudit [–types role_names] [–onlynames]
   [–list] [–server server_name] [–version] [–help]
```

Parameters

–types *role_names* – specifies the roles on which to get auditing information, where *role_names* lists one or more valid SQL Server roles. The valid roles are **sa_role**, **sso_role**, and **oper_role**. If the list of role names contains spaces, you must enclose the list in double quotation marks (see “Option Lists” on page 1-3). If you omit this option, **sgetroleaudit** displays auditing information for all three roles.

–onlynames – displays only the names of the roles being audited.

–list – displays output in list report format (see “Tabular and List Output Format Options” on page 1-10).

–server *server_name* – specifies the managed SQL Server for which you want the role auditing information, where *server_name* can be any valid managed SQL Server name (see “SQL Server Names” on page 1-15). The default is the current managed SQL Server collection (see “Command Context” on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. `sgetroleaudit`

   Displays the status of privileged command auditing for all roles in the current managed SQL Server:

```
*** Permission Auditing in Server: UMMG ***
Type           Value
-------------  -----
oper commands    Off
sa commands      Off
sso commands     Off
```

2. **`sgetroleaudit –list`**

   Displays summary of privileged command auditing for all roles in the current managed SQL Server. Output is in list format:

```
*** Permission Auditing in Server: UMMG ***
Permission Type: oper commands
Privileged Command Auditing: Off

Permission Type: sa commands
Privileged Command Auditing: Off

Permission Type: sso commands
Privileged Command Auditing: Off
```

3. **`sgetroleaudit –types sso_role –server UMMG`**

   Displays the status of privileged command auditing configuration for the **sso_role** in SQL Server UMMG:

```
*** Permission Auditing in Server: UMMG ***
Type          Value
------------  -----
sso commands    Off
```

Permissions

To use **sgetroleaudit**, you must have the following roles:

TME	ESSM	SQL Server
any	any	**sso_role**

Comment

If you do not supply any command line options, **sgetroleaudit** reports on the auditing status of all three roles in the current managed SQL Server.

See Also

scrtauditrec, sgetdbaudit, sgetloginaudit, sgetprocaudit, sgetserveraudit, sgettableaudit, sinstallaudit, ssetdbaudit, ssetloginaudit, ssetprocaudit, ssetroleaudit, ssetserveraudit, ssettableaudit

sgetrule

Function

Displays the rules for table columns.

Syntax

```
sgetrule [-names rule_names]
   [-onlynames | [ [-summary] [-bindings] ]]
   [-database db_path][-wildcard] [-version] [-help]
```

Parameters

–names *rule_names* – specifies the names of the rules to retrieve information on. The name of the rules can be prefixed with the name of the owner as follows: *owner.rule_name*. If you do not specify **–names**, the information is retrieved for all rules defined in the specified database.

–onlynames – displays only the names of the rules. Cannot be specified with **–bindings**.

–summary – displays summary information for the rules. Information includes the rule name, owner, creation date, and SQL for the rules. This is the default option if you do not specify **–bindings**. The output is in list mode only.

–bindings – displays the objects and table columns that these rules are bound to. Cannot be specified with **–onlynames**. The object name, type, and owner are displayed for each binding.

–database *db_path* – specifies the database in which the rules exist. The default is the current managed database.

–wildcard – enables wildcard pattern matching on the rule names. You can use wildcards for both the *owner* and *rule_name*, for example, dbo.%,%.rule1, rule%, and j%.r%.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
sgetrule -data ZURICH/pubs2
```

This command displays information for all rules located in the database *pubs2* in the managed SQL Server ZURICH.

```
*** Rules in ZURICH/pubs2 ***
Rule Name: pub_idrule
Owner Name: dbo
Rule Create Date: Oct 11 1995  4:26PM
Rule SQL:
    create rule pub_idrule
as @pub_id in ("1389", "0736", "0877", "1622",
"1756")
or @pub_id like "99[0-9][0-9]"

Rule Name: title_idrule
Owner Name: dbo
Rule Create Date: Oct 11 1995  4:26PM
Rule SQL:
    create rule title_idrule
as
@title_id like "BU[0-9][0-9][0-9][0-9]" or
@title_id like "[MT]C[0-9][0-9][0-9][0-9]" or
@title_id like "P[SC][0-9][0-9][0-9][0-9]" or
@title_id like "[A-Z][A-Z]xxxx" or
@title_id like "[A-Z][A-Z]yyyy"

/*valid values:  BU, MC,
     TC, PS, PC + 4 digits
**or
**any two uppercase letters followed by x's or y's
*/
```

Permissions

To use **sgetrule**, you must have the following roles:

TME	ESSM	SQL Server
any	any	any

See Also

scrtrule, sdelrule, ssetrule, scopyrule, scomprule

sgetseg

Function

Gets information about segments in a database. By default, **sgetseg** displays a summary report of the following attributes for each segment:

- Segment name
- Size in megabytes
- Used space in megabytes
- Free space in megabytes
- Percent full

You may also request information about:

- Device allocation for each segment
- Database objects in each segment
- Thresholds defined in each segment

Syntax

```
sgetseg [–names segment_names [–wildcard]]
   [–onlynames | [–allocation] [-devices] [–objects]
       [–thresholds] [–summary]]
   [–list] [–database db_path] [–version] [–help]
```

Parameters

–names *segment_names* – specifies one or more segments on which you want information. Without this option, **sgetseg** gets information on all segments in the database. If the list of segment names contains spaces, you must enclose the list in double quotation marks (see “Option Lists” on page 1-3).

–wildcard – enables wildcard matching on *segment_names* (see “Wildcards” on page 1-4).

–onlynames – displays only the names of the segments; the output is in list format.

–allocation – displays details about the devices and space allocated to each segment, including:

- Device name
- Starting virtual address
- Size in megabytes
- Reserved space in megabytes
- Free space in megabytes

–devices – specifies particular devices.

–objects – displays details about the database objects that each segment contains, including:

- Owner name
- Object name
- Index name
- Kind of object
- Type code of object
- Reserved space in megabytes

–thresholds – displays details about the thresholds defined on each segment, including:

- Page limit
- Percent full
- Procedure name

–summary – includes summary report attributes in the output (see “Attribute Summary and Detail Output Options” on page 1-11).

–list – displays output in list report format (see “Tabular and List Output Format Options” on page 1-10).

–database *db_path* – specifies the database in which the segments reside, where *db_path* can be any valid managed database name (see “Database Names” on page 1-16). The default is the current database collection (see “Command Context” on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`sgetseg`**

   Displays summary information for all segments in the current managed database:

```
*** Segments In Database: KOKO/pubs2 ***
Segment      Size  Used   Free   Percent
             in MB in MBin MBFilled
----------   ----  -----  -----  ------
default         2  1.172  0.828      59
logsegment      2  1.172  0.828      59
system          2  1.172  0.828      59
```

2. **`sgetseg -names default -list`**

   Displays summary information in list format for the *default* segment in the current managed database:

```
*** Segments In Database: KOKO/pubs2 ***
Segment Name: default
Size in Megabytes: 2
Reserved Space in Megabytes: 1.172
Free Space in Megabytes: 0.828
Percent full: 59
```

3. **`sgetseg –onlynames`**

   Displays only the names of the segments in the current managed database:

```
default
logsegment
system
```

4. **`sgetseg –allocation –database UMMG/pubs2`**

   Displays device allocation details for all segments in the *UMMG/pubs2* database. Display is in list format:

```
*** Segments In Database: UMMG/pubs2 ***
Segment Name: default
Size in Megabytes: 2
Reserved Space in Megabytes: 1.172
Free Space in Megabytes: 0.828
Percent full: 59
Allocation:
    Device Name: data_dev1
    Starting Virtual Address: 50331648
    Size in Megabytes: 2
    Reserved Space in Megabytes: 1.172
    Free Space in Megabytes: 0.828

Segment Name: logsegment
Size in Megabytes: 2
Reserved Space in Megabytes: 1.172
Free Space in Megabytes: 0.828
Percent full: 59
Allocation:
    Device Name: data_dev1
    Starting Virtual Address: 50331648
    Size in Megabytes: 2
    Reserved Space in Megabytes: 1.172
    Free Space in Megabytes: 0.828

Segment Name: system
Size in Megabytes: 2
Reserved Space in Megabytes: 1.172
Free Space in Megabytes: 0.828
Percent full: 59
Allocation:
    Device Name: data_dev1
    Starting Virtual Address: 50331648
    Size in Megabytes: 2
    Reserved Space in Megabytes: 1.172
    Free Space in Megabytes: 0.828
```

5. **`sgetseg –thresholds –database KOKO/pubs2`**

   Displays detailed threshold information for the segments in the *pubs2* database.

```
** Segments in KOKO/pubs2 ***

Segment Name: default
Size in Megabytes: 4
Reserved Space in Megabytes: 1.141
Free Space in Megabytes: 2.859
Percent Full: 29
Thresholds on Segment:

Segment Name: logsegment
Size in Megabytes: 4
Reserved Space in Megabytes: 1.141
Free Space in Megabytes: 2.859
Percent Full: 29
Thresholds on Segment:
    Page Limit: 104
    Percent Full: 94.922
    Procedure Name: sp_thresholdaction

Segment Name: system
Size in Megabytes: 4
Reserved Space in Megabytes: 1.141
Free Space in Megabytes: 2.859
Percent Full: 29
Thresholds on Segment:
```

Comment

If you do not supply any command line options, **sgetseg** reports on all segments in the current managed database.

Permissions

To use **sgetseg**, you must have the following roles:

TME	ESSM	SQL Server
any	any	**sso_role** or valid database user. To use the **–threshold** option, you must also have **sa_role**

See Also

scompseg, scopyseg, scrtseg, sdelseg, sgetdb, sgetdev, ssetseg

sgetserver

Function

Gets information about managed servers. By default, **sgetserver** displays a summary report of the following SQL Server attributes:

- SQL Server version
- Platform
- Operating system
- Build options
- Build date

You may also request information about:

- Configuration variable values
- SQL Server performance statistics
- Server status
- Languages installed

Syntax

```
sgetserver [–name server_name] [–config {"%" | names}]
   [–summary] [–stats] [–status] [–languages]
   [-list] [–version] [–help]
```

Parameters

–name *server_name* – specifies the SQL Server installation for which you want information, where *server_name* is any valid managed SQL Server name (see “SQL Server Names” on page 1-15). If you omit this option, **sgetserver** displays information for the current managed SQL Server.

–config {*“%”* | *names*} – specifies SQL Server configuration variables to display, along with their current values. If the list contains spaces, you must enclose the list in double quotation marks (see “Option Lists” on page 1-3).

The valid configuration variable names for SQL Server release 10.x are:

additional netmem
allow updates
audit queue size
cpu flush

database size
default character set id
default language
default network packet size
default sortorder id
devices
engine adjust interval
extent i/o buffers
fill factor
i/o flush
identity burning set factor
language in cache
locks
max online engines
maximum network packet size
memory
min online engines
nested triggers
open databases
open objects
password expiration interval
pre-read packets
procedure cache
recovery flags
recovery interval
remote access
remote connections
remote logins
remote sites
stack size
tape retention
time slice
upgrade version
user connections

The valid configuration variable names for SQL Server release 11.x are:

additional network memory
address lock spinlock ratio
allow nested triggers
allow remote access
allow sql server async i/o
allow updates to system tables
audit queue size
configuration file
cpu accounting flush interval
cpu grace time
deadlock checking period

deadlock retries
default character set id
default database size
default fill factor percent
default language id
default network packet size
default sortorder id
disk i/o structures
engine adjust interval
event buffers per engine
executable code size
freelock transfer block size
housekeeper free write percent
i/o accounting flush interval
i/o polling process count
identity burning set factor
identity grab size
lock promotion HWM
lock promotion LWM
lock promotion PCT
lock shared memory
max async i/os per engine
max async i/os per server
max engine freelocks
max network packet size
max number of network listeners
max online engines
memory alignment boundary
min online engines
number of alarms
number of devices
number of extent i/o buffers
number of index trips
number of languages in cache
number of locks
number of mailboxes
number of messages
number of oam trips
number of open databases
number of open objects
number of preallocated extents
number of remote connections
number of remote logins
number of remote sites
number of sort buffers
number of user connections
o/s asynch i/o enabled
o/s file descriptors

page lock spinlock ratio
page utilization percent
partition groups
partition spinlock ratio
permission cache entries
print deadlock information
print recovery information
procedure cache percent
recovery interval in minutes
remote server pre-read packets
runnable process search count
shared memory starting address
size of auto identity column
sort page count
sql server clock tick length
stack guard size
stack size
systemwide password expiration
table lock spinlock ratio
tape retention in days
tcp no delay
time slice
total data cache size
total memory
upgrade version
user log cache size
user log cache spinlock ratio

See Appendix A, "SQL Server Configuration Parameters" in *Enterprise SQL Server Manager User's Guide* for a description of these configuration variables. If you specify an option that contains spaces, you must enclose the option in single quotation marks. This option always accepts wildcard characters (%). See "Wildcards" on page 1-4.

–config *config_names* – reports the values of the configuration variables listed in *config_names*. By default, wildcard use is enabled within **–config** parameter entries.

The report includes:

- Configuration variable name
- Whether variable status is static or dynamic. Dynamic variables change when a reconfigure command is issued. Static variables take effect when you restart SQL Server.
- Minimum value
- Maximum value

- Current value
- Default value
- Comment

See Appendix A, "Configuration Variables" for additional information.

–stats – displays the following SQL Server performance statistics:

- Connections
- Packets sent
- Packet errors
- Packets received
- Total disk reads
- Total I/O errors
- Total disk writes
- Idle
- CPU busy
- I/O busy

Statistics values are for two time intervals: since SQL Server start-up and since the last execution of the **sgetserver** command.

–status – displays the current status of SQL Server. Status is one of the following:

ALIVE
DEAD

–languages – displays the languages installed on SQL Server.

–summary – includes summary report attributes in the output (see “Attribute Summary and Detail Output Options” on page 1-11).

–list – displays output in list report format (see “Tabular and List Output Format Options” on page 1-10).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`sgetserver -list`**

   Displays a summary report in list format for the current managed SQL Server:

```
*** Server KOKO ***
SQL Server Version: SQL Server 10.0.1
Platform: Sun4
Operating System: OS 4.1.x
Build Options: 1 OPT
Build Date: Tue Apr  5 03:38:51 PDT 1994
```

2. **`sgetserver –name UMMG –status –summary –list`**

   Displays status and release information in the managed SQL Server UMMG:

```
*** Server UMMG ***
Status: ALIVE
SQL Server Version: SQL Server 10.0.1
Platform: Sun4
Operating System: OS 4.1.x
Build Options: 1 OPT
Build Date: Tue Apr  5 03:38:51 PDT 1994
```

3. **`sgetserver –config "rec%, 'time slice'" –list`**

   Displays the current values of the configuration variables **recovery flags**, **recovery interval**, and **time slice** in the current managed SQL Server:

```
*** Server KOKO ***
Configuration Parameter: recovery flags
Default Configuration Value: 0
Current Configuration Value: 0
Minimum Value: 0
Maximum Value: 1
Configuration Status: 0
Configuration Comment: Recovery flags

Configuration Parameter: time slice
Default Configuration Value: 0
Current Configuration Value: 100
Minimum Value: 50
Maximum Value: 1000
Configuration Status: 0
Configuration Comment: Average time slice per
```

```
process in milliseconds

Configuration Parameter: recovery interval
Default Configuration Value: 3
Current Configuration Value: 3
Minimum Value: 1
Maximum Value: 32767
Configuration Status: 1
Configuration Comment: Maximum recovery interval
in minutes
```

4. sgetserver –stats

Displays the performance statistics for the current managed SQL Server. The report includes statistics gathered since SQL Server was last started and those gathered since the report was last requested by **sgetserver** or **sp_monitor**:

```
*** Server UMMG ***
Command Last Executed: 03/27/95 11:49:34 am
Current Command Executed: 03/27/95 11:51:07 am
Elapsed Time (sec): 93
Statistics:
    Statistic Name: Connections
    Since Server Startup: 656
    Since Last Execution: 0
    Percent: null

    Statistic Name: Packets Sent
    Since Server Startup: 6368
    Since Last Execution: 5
    Percent: null

    Statistic Name: Packet Errors
    Since Server Startup: 1
    Since Last Execution: 0
    Percent: null

    Statistic Name: Packets Received
    Since Server Startup: 5302
    Since Last Execution: 11
    Percent: null

    Statistic Name: Total Disk Reads
    Since Server Startup: 1913
    Since Last Execution: 0
    Percent: null

    Statistic Name: Total I/O Errors
    Since Server Startup: 0
    Since Last Execution: 0
    Percent: null

    Statistic Name: Total Disk Writes
    Since Server Startup: 177099
    Since Last Execution: 44
    Percent: null

    Statistic Name: Idle
    Since Server Startup: 583411
Since Last Execution: 93
Percent: 98

Statistic Name: CPU Busy
Since Server Startup: 823
Since Last Execution: 0
Percent: 0

Statistic Name: I/O Busy
Since Server Startup: 0
Since Last Execution: 0
Percent: 0
```

Comments

- Before using **sgetserver**, you must know the name of the managed SQL Server about which you want to retrieve information.
- **sgetserver** always evaluates wildcards in the **–config** list of configuration variables. Use [ ] to escape evaluation of wildcard characters in the **–config** list (see "Wildcards" on page 1-4). You can use wildcards to abbreviate configuration variable names, provided the string is unambiguous.
- This command does not display any information regarding the *runserver* file or *errorlog* file.

Permissions

To use **sgetserver**, you must have the following roles:

TME	ESSM	SQL Server
any	any	**sa_role**

See Also

scompserver, scopyserver, sgetprocess, skillprocess, sstartserver, sstopserver

sgetserveraudit

Function

Gets the status of SQL Server-wide auditing options (enabled or disabled). By default, **sgetserveraudit** displays a summary report for the status of the following SQL Server-wide auditing options:

- Ad hoc records
- System-wide auditing
- Fatal error auditing
- Login attempt auditing
- Logout auditing
- **set role** usage auditing
- Remote procedure call (RPC) auditing
- Server startups

Syntax

```
sgetserveraudit [-name server_name] [-list]
   [–version] [–help]
```

Parameters

–name *server_name* – specifies the managed SQL Server on which to get auditing information, where *server_name* can be any valid managed SQL Server name (see “SQL Server Names” on page 1-15). The default is the current managed SQL Server collection (see “Command Context” on page 1-15).

–list – formats the output as a list.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
sgetserveraudit -name UMMG
```

Displays the status of SQL Server-wide auditing options for managed SQL Server UMMG:

```
*** Auditing in Server: UMMG ***
Type             Value
---------------  -----
adhoc records      Off
enable auditing    Off
errors             Off
logins             Off
logouts            Off
roles              Off
rpc connections    Off
server boots       Off
```

Comment

If you do not supply any command line options, **sgetserveraudit** reports about the current managed SQL Server.

Permissions

To use **sgetserveraudit**, you must have the following roles:

TME	ESSM	SQL Server
any	any	**sso_role**

See Also

scrtauditrec, sgetdbaudit, sgetloginaudit, sgetprocaudit, sgetroleaudit, sgettableaudit, sinstallaudit, ssetdbaudit, ssetloginaudit, ssetprocaudit, ssetroleaudit, ssetserveraudit, ssettableaudit

sgetsybaselogin

Function

Returns information about the SQL Server login set for a specific ESSM administrator. Passwords are not displayed.

Syntax

sgetsybaselogin –principal *username* [–version] [–help]

Parameters

–principal *username* – specifies the Tivoli name for the user. On UNIX systems, this is the UNIX login name.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **sgetsybaselogin –principal webster**

   Displays information about the SQL Server login account for administrator *webster* in the TMR:

   ```
   SQL Server login information for user: webster
   -- Enterprise-wide Information --
   Enterprise-wide login name: webster
   Enterprise-wide login password: <encrypted>

   -- Policy Region Login Information --
   None

   -- SQL Server Login Information --
   None
   ```

2. **sgetsybaselogin -principal harryc**

   Displays information about the SQL Server login account for administrator *harryc* in the TMR:

```
SQL Server login information for user: harryc
-- Enterprise-wide Information --
Enterprise-wide login name: harryc
Enterprise-wide login password: <encrypted>

-- Policy Region Login Information --
None

-- SQL Server Login Information --
SQL Server name: BOSTON
SQL Server login name: harryc
SQL Server login password: <encrypted>
```

Comment

You can use the **-sybasepassword** option of the **ssetlogin** command to coordinate changes in the administrator's login and SQL Server login account.

Permissions

To use **sgetsybaselogin**, you must have the following roles:

TME	ESSM	SQL Server
super	none	none

See Also

scrtlogin, scrtsybaselogin, sdelsybaselogin, sgetlogin, ssetlogin, ssetsybaselogin

sgettable

Function

Displays the definition of tables.

Syntax

```
sgettable [-names table_names]
   [-onlynames | [ [-summary] [-columns] [-triggers]
       [-indexes] [-bindings] [-referencedby]
       [-list] ]]
   [-database db_path] [-wildcard] [-version] [-help]
```

Parameters

–names *table_names* – specifies the names of the tables to retrieve information on. The names of the tables can be prefixed with the name of the owner as follows: *owner.table_name*. If you do not specify **–names**, the information is retrieved for all tables defined in the specified database.

–onlynames – displays only the names of the tables.

–summary – displays summary information for the tables. Information includes the table name, owner, current segment that the table resides on, and creation date for the tables. **–summary** is the default parameter if you do not specify **–columns**, **–indexes**, **–triggers**, **–spaceused**, **–bindings** or **–referencedby.**

–columns – displays the column definitions for each table. The output is in **–list** mode. The following information is displayed for each column:

- datatype name
- type
- length
- precision
- scale
- allow nulls setting

–triggers – displays the table trigger information. The output is in **–list** mode. **–triggers** displays the trigger name, type, owner, and text for each trigger.

–indexes – displays the table index information. The output is in **–list** mode. The index name, type, and owner are displayed for each index.

–bindings – displays the list of objects – defaults, rules, and datatypes – that are bound to this table. The output is in **–list** mode. The object name, column name, type, and owner are displayed for each binding.

–referencedby – displays the stored procedures, triggers and views that reference this table. The output is in **–list** mode. The object name, type, and owner are displayed for each reference.

–list – displays the output in list format.

–database *db_path* – specifies the database where the tables reside. The default is the current managed database.

–wildcard – enables wildcard pattern matching on the table names.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
sgettable -name salary_table -summary -columns
   -database PARIS/hrdb
```

This command displays information about the table *salary_table* in the database *hrdb,* in the managed SQL Server PARIS. The command displays summary information and information about the columns in the table.

```
** Tables in PARIS/hrdb **
Name: salary_table
Owner: dbo
Segment: my_seg
Creation Date: Feb 7 1995 2:34PM
Columns:

Name: ssn
Type: varchar
Length: 11
Precision: 0
Scale: 0
Allow Nulls: No
```

```
Name: name
Type: varchar
Length: 50
Precision: 0
Scale: 0
Allows Nulls: No

Name: salary
Type: long
Length: 1
Precision: 0
Scale: 0
Allow Nulls: No
```

Permissions

To use **sgettable**, you must have the following roles:

TME	ESSM	SQL Server
any	any	any

See Also

scrttable, sdeltable, ssettable, scopytable, scomptable, schecktable

sgettableaudit

Function

Gets status of command auditing for tables and views in a database. By default, **sgettableaudit** displays a summary report showing the auditing status for usage attempts of the following commands in each table or view:

- **delete**
- **insert**
- **select**
- **update**

For each command, **sgettableaudit** shows the auditing status of successful usage attempts, failed usage attempts, both, or neither. Optionally, **sgettableaudit** also reports the default command auditing that will be applied to all new tables and views.

Syntax

```
sgettableaudit [–names tables_and_views [–wildcard]]
   [–new] [–onlynames] [-list] [–database db_path]
   [–version] [–help]
```

Parameters

–names *tables_and_views* – specifies which tables and views to get auditing information about, where *tables_and_views* lists one or more existing table or view names in the database. If the list contains spaces, you must enclose it in double quotation marks (see “Option Lists” on page 1-3). If you omit this option, **sgettableaudit** displays auditing information for all stored procedures and triggers in SQL Server.

–wildcard – enables wildcard matching on *tables_and_views* (see “Wildcards” on page 1-4).

–new – displays (in list format) the default command auditing that will be applied to all new tables and views.

–onlynames – displays only the names of audited tables and views; the output is in list format.

–list – displays output in list report format (see “Tabular and List Output Format Options” on page 1-10).

–database *db_path* – specifies the database in which the audited tables and view reside, where *db_path* can be any valid managed database name (see "Database Names" on page 1-16). The default is the current database collection (see "Command Context" on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`sgettableaudit –names titleauthor –database AGRA/pubs2`**

   Displays the auditing status of the *titleauthor* table in the database *pubs2,* in the managed SQL Server AGRA:

```
*** Table/View Auditing in Database: AGRA/pubs2 ***
Name             Delete           Update  Select  Insert
---------------  ---------------  ------  ------  ------
titleauthor      Off              Off     Off     Off
```

2. **`sgettableaudit –names titl% -wildcard –new`**

   Displays the current auditing status for all tables and views beginning with "titl" in the database and the default auditing status for all new tables and views:

```
*** Table/View Auditing in Database: AGRA/pubs2 ***
Table/View Name: titleauthor
DELETE: Off
UPDATE: Off
SELECT: Off
INSERT: Off

Table/View Name: titles
DELETE: Success, Failure
UPDATE: Off
SELECT: Off
INSERT: Off

Table/View Name: titleview
DELETE: Off
UPDATE: Off
SELECT: Off
INSERT: Off

Table/View Name: future tables
```

```
DELETE: Off
UPDATE: Off
SELECT: Off
INSERT: Off

Table/View Name: future views
DELETE: Off
UPDATE: Off
SELECT: Off
INSERT: Off
```

Comment

If you do not supply any command line options, **sgettableaudit** reports on all tables and views in the current database.

Permissions

To use **sgettableaudit**, you must have the following roles:

TME	ESSM	SQL Server
any	any	**sso_role** or **sa_role**

See Also

scrtauditrec, sgetdbaudit, sgetloginaudit, sgetprocaudit, sgetroleaudit, sgetserveraudit, sinstallaudit, ssetdbaudit, ssetloginaudit, ssetprocaudit, ssetroleaudit, ssetserveraudit, ssettableaudit

sgettrigger

Function

Displays information about SQL Server triggers in a database.

Syntax

```
sgettrigger [-names trigger_names]
   [-onlynames |[ [-summary] [-referencedby] ]]
   [-database db_path] [-wildcard] [-version] [-help]
```

Parameters

–names *trigger_names* – specifies the names of the triggers to retrieve information on. The names of the triggers can be prefixed with the name of the owner as follows: *owner.trigger_name*. If you do not use **–names**, the information is retrieved for all triggers defined in the specified database.

–onlynames – displays only the names of the triggers.

–summary – displays summary information for the triggers. Information includes the trigger name, table name, owner, creation date, and SQL. This is the default option if you do not specify **–referencedby.**

–referencedby – displays the list of objects – stored procedures, tables, and views – that the triggers depend on. **–referencedby** displays object name, type, and owner for each reference.

–database *db_path* – specifies the database in which the triggers exist. The default is the current managed database.

–wildcard – enables wildcard pattern matching on the trigger names.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

➤ ***Note***

This command displays in list mode only. There is no tabular output.

Example

```
sgettrigger -name deltitle -data BOSTON/pubs2
```

This command displays information about the trigger *deltitle* in the database *pubs2*, in the managed SQL Server BOSTON.

```
*** Triggers in Database: JOHNNYP/pubs2 ***
Trigger Name: deltitle
Owner Name: dbo
Trigger Create Date: Jul  3 1995  3:20PM
Trigger SQL:
/*** Create some standard triggers */

create trigger deltitle
on titles
for delete
as
    if (select count(*) from deleted, salesdetail
    where salesdetail.title_id = deleted.title_id)
>0
    begin
        rollback transaction
        print "You can't delete a title with
sales."
end
```

Permissions

To use **sgettrigger**, you must have the following roles:

TME	ESSM	SQL Server
any	any	any

See Also

scrttrigger, sdeltrigger, scopytrigger, scomptrigger

sgetuser

Function

Gets information about users that have access to a specific SQL Server database. By default, the **sgetuser** command displays a summary report of the following attributes:

- User name
- Login name
- Group
- Database ownership

Syntax

```
sgetuser [–names user_names [–wildcard]]
   [–onlynames | [–summary] [–list] [–aliases]]
   [–database db_path] [–version] [–help]
```

Parameters

–names *user_names* – restricts the summary report to the user names listed. Omitting this optional parameter causes all of the database users to be included in the summary report.

–wildcard – enables wildcard matching on *user_names* (see “Wildcards” on page 1-4).

–onlynames – requests all database user names in list format.

–summary – requests summary report attributes.

–list – requests the list report format.

–aliases – requests user alias information in the report.

–database *db_path* – specifies the database in which the users reside, where *db_path* can be any valid managed database name (see “Database Names” on page 1-16). The default is the current database collection (see “Command Context” on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`sgetuser –database KOKO/pubs2 –names webster –alias –list`**

   Displays *pubs2* database user access information for the user *webster* in the managed SQL Server KOKO.

```
*** Users in Database: KOKO/pubs2 ***
User Name in Database: webster
Aliases:
    Login Name: barneyb
    Login Name: harryc
    Login Name: probe
```

2. **`sgetuser –database pubs2`**

   Displays all database user access information for the *pubs2* database in the current managed SQL Server.

```
*** Users in Database: KOKO/pubs2 ***
User     Login     Group
-------  --------  ------
guest    null      public
webster  bwebster  public
```

3. **`sgetuser –database KOKO/pubs2 –onlynames`**

   Displays the names of all users with access to the *pubs2* database in the managed SQL Server KOKO.

```
guest
webster
```

Comment

To use **sgetuser**, you must know the name of the database whose user attributes you want reported.

Permissions

To use **sgetuser**, you must have the following roles:

TME	ESSM	SQL Server
any	any	**sso_role**, **sa_role**, or valid database user

See Also

scrtuser, sdeluser, sgetgroup, ssetuser

sgetview

Function

Displays definitions of views.

Syntax

```
sgetview [-names view_names]
   [-onlynames |
       [[-summary] [-bindings] [-referencedby]]]
   [-database db_path] [-wildcard] [-version] [-help]
```

Parameters

–names *view_names* – specifies the names of the views to retrieve information on. The names of the views can be prefixed with the name of the owner as follows: *owner.view_name*. If you do not specify **–names**, the information is retrieved for all views defined in the specified database.

–onlynames – displays only the names of the views.

–summary – displays summary information for the views. Information includes the view name, owner, creation date, and SQL for the views. This is the default option if you do not specify **–columns**, **–bindings** or **–referencedby.**

–bindings – displays the list of objects: tables and views that these views depends on. The object name, type, and owner are displayed for each binding.

–referencedby – displays the stored procedures, triggers, and views that reference these views. The object name, type, and owner are displayed for each reference.

–database *db_path* – specifies the database in which the views exist. The default is the current managed database.

–wildcard – enables wildcard pattern matching on the view names.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

This command always displays the output in list mode.

Example

```
sgetview -data BOSTON/pubs2 -name titleview
```

This command displays the definition of the view *titleview*, in the database *pubs2*, in the managed SQL Server BOSTON.

```
*** Views in JOHNNYP_DEV/pubs2 ***
View Name: titleview
Owner Name: dbo
View Create Date: Nov  6 1995  2:34PM
Summary Text:
/*** Create a few views now.  Start with a view
combining the authors, titles, and titleauthors
tables***/
create view titleview
as
        select title, au_ord, au_lname,
        price, total_sales, pub_id
        from authors, titles, titleauthor
        where authors.au_id = titleauthor.au_id
        and titles.title_id = titleauthor.title_id
```

Permissions

To use **sgetview**, you must have the following roles:

TME	ESSM	SQL Server
any	any	any

See Also

scrtview, sdelview, scopyview, scompview

sinstallaudit

Function

Installs the audit system in a managed SQL Server.

Syntax

```
sinstallaudit [-server server_name]
       -dbsize auditing_db_size
       -physical physical_name
       [-name auditing_device_name]
       [-size auditing_device_size]
       [–version] [–help]
```

Parameters

–server *server_name* – specifies the managed SQL Server in which to install the auditing system. The default is the current managed SQL Server collection (see “Command Context” on page 1-15).

–dbsize *auditing_db_size* – specifies the size (in megabytes) of the *sybsecurity* database that is created when auditing is installed.

–physical *physical_name* – specifies the physical name of the device on which to install auditing. If this device does not exist, it will be created.

–name *auditing_device_name* – specifies the logical name of the auditing device. The default auditing device name is *sybsecurity.*

–size *auditing_device_size* – specifies the size (in megabytes) of the device that will be created for the auditing database. The default is 5MB.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. `sinstallaudit -server BURL -dbsize 5 -physical dev1`

   Installs auditing on SQL Server BURL, specifying 5MB as the size of the *sybsecurity* database. The physical device on which to install auditing is *dev1*. The default values for logical name of the auditing device and the auditing database size are used.

2. **`sinstallaudit -server BURL -dbsize 4`**
   `-physical dev1 -name d01 -size 4`

   Installs auditing on SQL Server BURL, specifying 4MB as the size of the *sybsecurity* database. The physical device on which to install auditing is *dev1*. The logical name of the auditing device is *d01* and the auditing database size is 4MB.

Comments

- Before using **sinstallaudit**, you need the name of the SQL Server in which to install the auditing system.
- The SQL Server installation you are auditing must exist on a node that is a managed node within the TME.
- To install auditing, ESSM creates the *sybsecurity* database and creates several stored procedures (in the *sybsystemprocs* database) that help administer the auditing configuration.

Permissions

To use **sinstallaudit**, you must have the following roles:

TME	ESSM	SQL Server
any	**server**	**sso_role** and **sa_role**

See Also

scrtauditrec, sgetdbaudit, sgetloginaudit, sgetprocaudit, sgetroleaudit, sgetserveraudit, sgettableaudit, ssetdbaudit, ssetloginaudit, ssetprocaudit, ssetroleaudit, ssetserveraudit, ssettableaudit

skillprocess

Function

Kills a process running on a managed SQL Server.

Syntax

```
skillprocess –id process_id [–server server_name]
   [–version] [–help]
```

Parameters

–id *process_id* – specifies the ID of the process to kill.

–server *server_name* – specifies the managed SQL Server running the process. The default is the current managed SQL Server collection (see “Command Context” on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
skillprocess –id 101 –server BURL
```

Kills the process with a process ID of 101 on the SQL Server BURL.

Comments

- Before using **skillprocess**, you need the following information:
  - The name of the managed SQL Server running the process.
  - The ID of the process you wish to kill.
- Processes that you cannot kill include the network handler, the checkpoint, and the current command.

Permissions

To use **skillprocess**, you must have the following roles:

TME	ESSM	SQL Server
any	**server**	**sa_role**

See Also

sgetserver, sgetprocess, sstartserver, sstopserver

sloaddb

Function

Restores a data or transaction log portion of a database to a specified managed SQL Server.

Syntax

```
sloaddb -name database [-server server] [-transaction]
   [-density density] [-blocksize blocksize]
   [-dumpvolume dump_volume_name] [-file file_name]
   [-nodismount] [-unload] [-listonly] [-headeronly]
   [-version] [-help]
```

Stripe device information must be specified via standard input and must contain one or more stripe device specifications, as follows:

```
-stripedevice device_name [-backupserver server_name]
   [-dumpvolume volume_name] [-file file_name]
   [-density density] [-blocksize blocksize]
```

Parameters

–name *database* – specifies the database to restore.

–server *server* – specifies the SQL Server installation to which the database will be restored.

–transaction – loads a backup copy of the transaction log.

–density *density* – overrides the default density for a tape with a specified density.

–blocksize *blocksize* – overrides the default blocksize for a dump device with a specified blocksize.

–dumpvolume *dump_volume_name* – specifies the name of the volume containing the database.

–file *file_name* – specifies the name of the dump file.

–nodismount – specifies that the tape is not dismounted after the database is restored.

–unload – rewinds the tape when the database is restored.

–listonly – displays information about all dump files on the tape, but does not restore the database.

–headeronly – displays header information for a single dump file, but does not restore the database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

–stripedevice *device_name* – specifies the name of the stripe device.

–backupserver *server_name* – specifies the name of the backup SQL Server.

–loadvolume *volume_name* – specifies the volume.

–file *file_name* – specifies the name of the dump file.

–density *density* – overrides the default density for a tape with a specified density.

–blocksize *blocksize* – overrides the default blocksize for a dump device with a specified blocksize.

Example

```
sloaddb -name pubs2 -server BOSTON <stripe.opt
```

where the file *stripe.opt* contains the following:

```
-stripedevice sdev1 -backupserver BACKUP
-loadvolume pvolume
```

Restores the *pubs2* database to BOSTON from dump device *sdev1* in BACKUP.

Comments

- Before you use the **sloaddb** command, you need to know:
  - The name of the database to restore
  - The name and location of the dump device from which the data or log is being restored
- Restoration messages are displayed in the TME Administrator notice group.

Permissions

To use **sloaddb**, you must have the following roles:

TME	**ESSM**	**SQL Server**
admin, **senior**, or **super**	**load**	**sa_role**

See Also

schangevol, scrtdumpdev, scrtmirror, sdumpdb, smanageserver

smanageserver

Function

Registers SQL Server as a **managed resource** in the TMR. Before you can use ESSM to administer SQL Server and its objects, you must register SQL Server using **smanageserver**.

Syntax

```
smanageserver –name server_name –host host_name
   -policyregion policy_region
   [-management_host management_host_name]
   [-errorlog filename] [–version] [–help]
```

Parameters

–name *server_name* – the name of the SQL Server installation that you want to administer by Enterprise SQL Server Manager.

–host *host_name* – identifies the SQL Server host.

–policyregion *policy_region* – specifies the policy region of which the named Server is to become a managed resource.

–management_host *management_host_name* – identifies the machine on which ESSM is located and from which you will manage SQL Server. The default is the SQL Server host specified by **–host**.

–errorlog *filename* – specifies the full pathname of the SQL Server error log. Event Monitoring Services use this information to monitor the error log. The error log must be on the management host.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
smanageserver –name KOKO -policyregion RCA
–host redbud
```

Registers the SQL Server KOKO as a managed SQL Server in the policy region *RCA* on the management host *redbud*.

Comments

Before using **smanageserver**, you must know:

- the name of the SQL Server object that you are registering as a managed SQL Server
- the name of the policy region in which the Server will be a managed resource
- the host from which SQL Server will be managed

Permissions

To use **smanageserver**, you must have the following roles:

TME	ESSM	SQL Server
server	none	none

See Also

sunmanageserver, ssetserver, sgetserver, sstartserver, sstopserver

spopulateprf

Function

Adds names to and removes names from a specified ESSM profile of a specified type.

Syntax

spopulateprf -name *profile_name* -type *profile_type*
{{-addmembers *members* | -addall} |
-dropmembers *members*} [-version] [-help]

Parameters

–name *profile_name* – specifies the name of the profile.

–type *profile_type* – specifies the type of profile. ESSM profile types are classified as SQL Server profile types and database profile types. The SQL Server profile types are:

- SQLServerProfile
- SQLLoginProfile
- SQLDumpDeviceProfile
- SQLDbDeviceProfile
- SQLRemoteServerProfile
- SQLDatabaseProfile
- SQLCacheProfile

The database profile types are:

- SQLUserProfile
- SQLGroupProfile
- SQLSegmentProfile
- SQLDatabaseProfile
- SQLDefaultProfile
- SQLRuleProfile
- SQLDataTypeProfile
- SQLTableProfile
- SQLIndexProfile
- SQLViewProfile

- SQLProcedureProfile
- SQLTriggerProfile

–addmembers *members* – specifies a list of names to add to the profile.

–addall – Causes the profile to be populated with any elements in the associated SQL Server or database that don't already exist in a profile. For example, you have a database with five tables, all of which are in a table profile. Someone creates two additional tables through some tool other than ESSM. If you invoke this command on the table profile with the **-addall** option, the two new tables are added to the table profile.

–dropmembers *members* – specifies a list of names to remove from the profile.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
spopulateprf -name login_profile -type
SQLLoginProfile -addmembers "dianne, scott, sue"
-dropmembers "pat, jackie"
```

Adds login names to and removes login names from the profile called *login_profile*.

Comments

- The names that are added to and removed from a profile must existing in the SQL Server installation or database associated with the profile. For example, if you add the login, *smith*, to the login profile, *my_logins*, in SQL Server BOSTON, login *smith* must exist in BOSTON.
- The names are used as keys to retrieve information from the SQL Server installation or database associated with the profile manager.
- Removing a name from a profile does not affect the object represented by the name.
- The same name cannot be in two different profiles in the same profile manager.

- You must specify at least one of the options **–addmembers**, **–addall**, or **–dropmembers.** You can add members and drop members in the same command.

Permissions

To use **spopulateprf**, you must have the following roles:

TME	ESSM	SQL Server
senior	**server**; for database objects you also need **schema**	none

See Also

scrtprfmgr, sgetprf

ssetcmdperm

Function

Grants or revokes permission to use the following commands:

- **create database**
- **create default**
- **create procedure**
- **create rule**
- **create table**
- **create view**
- **dump database**
- **dump transaction**

Syntax

```
ssetcmdperm {-grant | -revoke}
   [-commands {command_names | all}]
   {{-users user_names | -groups group_names |
       -roles role_names}...}
   [-database db_path] [-version] [-help]
```

Parameters

–grant – grants permission on specified commands to specified users, groups, or roles.

–revoke – revokes permission on specified commands from specified users, groups, or roles.

–commands *command_names* – specifies the commands for which you are configuring permissions. Because the commands include spaces, enclose each command in quotation marks (‘). Also enclose the whole list in double quotation marks. See “Option Lists” on page 1-3 for more on specifying option lists with embedded spaces. The default is **all**.

all – configures permissions for all commands.

–users *user_names* – specifies the users for which you are configuring command permissions. If the list contains spaces, enclose the whole list in double quotation marks (see “Option Lists” on page 1-3).

–groups *group_names* – specifies the groups for which you are configuring command permissions. If the list contains spaces, enclose the whole list in double quotation marks (see “Option Lists” on page 1-3).

–roles *role_names* – specifies the roles for which you are configuring command permissions. If the list contains spaces, enclose the whole list in double quotation marks (see “Option Lists” on page 1-3).

–database *db_path* – specifies the database on which to get permission information. The default is the current database collection (see “Command Context” on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. `ssetcmdperm -user bwebster -revoke -command "'create rule'" -database AGRA/pubs2`

   Revokes the **create rule** command permission from *bwebster* in the *pubs2* database.

2. `ssetcmdperm -user john -grant -command "'create rule', 'create table'" -database AGRA/pubs2`

   Grants the **create rule** and **create table** command permissions to *john* in the *pubs2* database.

Comments

- Before using **ssetcmdperm**, you need the name of the database to access.
- Permission to use the **create database** command can be granted only by a System Administrator and only to users in the *master* database.
- You must use at least one **–users**, **–groups**, or **–roles** argument. You can use more than one.

Permissions

To use **ssetcmdperm**, you must have the following roles:

TME	ESSM	SQL Server
any	**security**	**sa_role** or Database Owner

See Also

sgetcmdperm, sgetobjperm, ssetobjperm

ssetcolumn

Function

Changes the datatype of a column on a table.

Syntax

```
ssetcolumn -name column_name -table table_name
   -type type [-length length] [-precision precision]
   [-scale scale] [-nulls | -nonulls | -identity] [-
   database db_path] [-version] [-help]
```

Parameters

–name *column_name* – specifies the name of the column to modify.

–table *table_name* – specifies the name of the table in which to modify the column. The name of the table can be prefixed with the name of the owner as follows: *owner.table_name*.

–type *type* – specifies the new type of the column.

–length *length* – specifies the length of the char, varchar, nchar, nvarchar, binary, and varbinary types. If you do not specify **–length**, and one of these types is used, the SQL Server default is 1.

–precision *precision* – The float, numeric, and decimal types require a precision. Float defaults to a platform-specific value, numeric and decimal defaults to 18.

–scale *scale* – Both numeric and decimal types also require a scale value. The sql server default is 0.

–nulls - specifies that the column allows null values. The default is **–nonulls**.

–nonulls – specifies that the column does not allow null values. **–nonulls** is the default.

–identity – specifies that the column contains a system generated, sequential value that identifies each row in the table. The default is **–nonulls**. Only one column can be specified with the **–identity** option.

–database *db_path* – specifies the database in which the table exists. The default is the current managed database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example:

```
ssetcolumn -name transaction_description
-type varchar -length 100 -table trades_table -
data NYSE/daily_db
```

This example changes the column *transaction_description* in the table *trades_table*, in the database *NYSE/daily_db*. It changes the type to *varchar,* with a length of 100.

Comment

Before you change the properties of a table column, you need to be sure that you have enough disk space to complete the procedure. Before Enterprise SQL Server Manager changes a column, it renames the target table. This allows Enterprise SQL Server Manager to restore the original table if the procedure fails. Therefore, during the process, you need to have enough space in the database to hold a copy of all tables being modified. If no errors occur during the table column modify operation, the renamed table is deleted.

Enterprise SQL Server Manager uses the Bulk Copy (BCP) utility to export and import the data from the source table. By default, the BCP files for either operation are stored in the *$DBDIR* of the management host for the source SQL Server. *$DBDIR* is the location of the Tivoli database. The BCP files are then distributed (copied) to the $DBDIR of the management host for the target. You must have enough disk space in both the source and target *$DBDIR* to accommodate these BCP files. As a rule, make sure the *$DBDIR* has free disk space equivalent to 1.5 times the size of the tables being modified.

If *$DBDIR* is not large enough, you can use an environment variable to specify an alternate location for the BCP files. The root user must have write permission in the directory you specify. The directory must exist on the source and target management hosts.

To modify table column properties, set the variable in the oserv environment of the management host of the source SQL Server. The variable is *ESSM_STAGING_DIR*.

To set *ESSM_STAGING_DIR* on the management host:

1. Retrieve the current environment definition on the management host. Save the output to a file.

   ```
   odadmin environ get > /tmp/odadmin.dat
   ```

2. Edit the output file. Specify a location for staging files, for example:

   ```
   ESSM_STAGING_DIR=/work/essm_staging
   ```

3. Save the changes to the output file.
4. Reset the oserv environment:

   ```
   odadmin environ set < /tmp/odadmin.dat
   ```

Permissions

To use **ssetcolumn**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	Database Owner

ssetdatatype

Function

Changes the bindings of a datatype.

Syntax

```
ssetdatatype -name datatype_name
   {[[-bindrule rule_name] | -unbindrule] |
       [[-binddefault default_name] | -unbinddefault]
       [-future]}
   [-database db_path] [-version] [-help]
```

Parameters

–name *datatype_name* – specifies the name of the user datatype to change.

–bindrule *rule_name* – specifies the name of a rule to bind to this datatype. If you specify a rule, it replaces any rule already specified for this datatype.

–unbindrule – removes the rule bound to this datatype.

-binddefault *default_name* – specifies the name of the default value to bind to this datatype. If you specify a default, it replaces any default already specified for this datatype.

–unbinddefault – removes the default bound to this datatype.

–future – prevents existing columns of the user datatype from acquiring (**–bind**), or losing (**–unbind**) the default or rule. Can be specified only with **–bindrule**, **–unbindrule**, **–binddefault**, or **–unbinddefault.**

–database *db_path* – specifies the database in which the datatype is located. The default is the current managed database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
ssetdatatype -name ssn -bindrule quiz_answer -
   database HRSERVER/hrdb
```

This example changes the user datatype *ssn* in the database *hrdb,* in the managed SQL Server HRSERVER by binding to it a rule named *quiz_answer.*

Permissions

To use **ssetdatatype**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	type owner

See Also

sgetdatatype, scrtdatatype, sdeldatatype, scopydatatype, scompdatatype

ssetdb

Function

Sets the configuration of a database in a SQL Server, including its ownership, devices, space allocations, and configuration options.

Syntax

```
ssetdb –name db_name [–owner owner_name]
   [–adddatadevices "{default | device} size
       [, device size]..." ]
   [–addlogdevices "{default | device} size
       [, device size]..." ]
   [–config "db_option Boolean
       [, db_option Boolean]..." ]
   [–override] [–forload] [–server server_name]
   [–version] [–help]
```

Parameters

–name *db_name* – specifies the name of the database to modify, where *db_name* is the name of an existing database. Collection path names are invalid here (use **–server** to specify a SQL Server collection object).

–owner *owner_name* – specifies the name of a new Database Owner, where *owner_name* is an existing SQL Server login name.

–adddatadevices – specifies additional database devices (**default** or *device*) and the corresponding space (*size*) on each device to allocate to the database for storing data. You must enclose the list of *device size* pairs in double quotation marks (see “Option Lists” on page 1-3).

default – indicates that the database can allocate additional space (specified by *size*) on any of the default database devices (listed by **sgetdev**).

device – specifies the logical name of an additional device in which the database may store information. A database can occupy different amounts of space on each of several database devices; therefore, you can specify multiple *device size* pairs in the argument list.

size – specifies the amount of additional space (in MB) to allocate on the specified device (**default** or *device*).

–addlogdevices – specifies the additional database devices (**default** or *device*) and the corresponding space (*size*) on each device to allocate to the database for storing the transaction log. You must enclose the list of *device size* pairs in double quotation marks (see “Option Lists” on page 1-3).

–config – enables or disables one or more database configuration options (*db_option*). This argument accepts a list of *db_option* {TRUE | FALSE} pairs. Because the argument list always contains spaces, you must enclose it in double quotation marks (see “Option Lists” on page 1-3).

db_option – specifies a database configuration option to enable or disable. The valid database option names are:

abort tran on log full
allow nulls by default
auto identity
dbo use only
ddl in tran
no chkpt on recovery
no free space acctg
read only
select into/bulkcopy
single user
trunc log on chkpt

See Appendix A, “Database Options” for a description of these database options. SQL Server accepts any unique string that is part of the option name. If you specify an option that contains spaces, you must enclose the option in single quotation marks.

Boolean – SQL Server Boolean keyword that enables or disables the preceding *db_option*. TRUE enables the option; FALSE disables the option. Because these Boolean argument values are interpreted by SQL Server as keywords, only the values TRUE and FALSE are valid (see “Boolean Keywords” on page 1-3).

–override – allows you to specify the same device name for both the data and the transaction log. This option allows SQL Server, on a machine with limited space, to still maintain the transaction log on separate device fragments from the database’s data. Without this option, the **ssetdb** command fails if you attempt to assign the data and transaction log to the same device.

–forload – specifies that the database is for use only in loading a database dump. Use this option to recover from a media failure or to move a database from one machine to another.

–server *server_name* – specifies the managed SQL Server in which the database (*db_name*) resides, where *server_name* can be any valid managed SQL Server name (see "SQL Server Names" on page 1-15). The default is the current managed SQL Server collection (see "Command Context" on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`ssetdb –name inventory –owner mike`**

   Changes the owner of the *inventory* database in the current managed SQL Server to *mike*.

2. **`ssetdb –name inventory –adddatadevices "DEFAULT 2" –server NEWYORK`**

   Increases the size of the *inventory* database in the managed SQL Server NEWYORK by 2 MB. Space is allocated on a default database device.

3. **`ssetdb –name pubs2 –config "’read only’ TRUE, ’dbo use only’ FALSE"`**

   Sets the **read only** database option to TRUE and the **dbo use only** configuration option to FALSE for the *pubs2* database.

Comments

- Before using **ssetdb**, you must know the name of the database to modify.
- By default, wildcard use is enabled within **–config** parameter entries.
- You cannot shrink a database.
- You can configure database options for all new databases by executing this command with the **–config** option on the database named *model*.
- You cannot change the database configuration options for the master database.

Permissions

To use **ssetdb**, you must have the following roles:

TME	ESSM	SQL Server
user	space	sa_role

See Also

scrtdb, scheckdb, scompdb, scopydb, sdeldb, sgetdb, ssetdev

ssetdbaudit

Function

Configures command auditing within one or more databases in a particular managed SQL Server.

Syntax

```
ssetdbaudit [-names db_names [-wildcard]]
   [-drop "{+|-}{SUCCESS|FAILURE}
       [{+|-}{SUCCESS|FAILURE}]"]
   [-use "{+|-}{SUCCESS|FAILURE}
       [{+|-}{SUCCESS|FAILURE}]"]
   [-truncatetable "{+|-}{SUCCESS|FAILURE}
       [{+|-}{SUCCESS|FAILURE}]"]
   [-grant "{+|-}{SUCCESS|FAILURE}
       [{+|-}{SUCCESS|FAILURE}]"]
   [-revoke "{+|-}{SUCCESS|FAILURE}
       [{+|-}{SUCCESS|FAILURE}]"]
   [-access "{+|-}{SUCCESS|FAILURE}
       [{+|-}{SUCCESS|FAILURE}]"]
   [-all "{+|-}{SUCCESS|FAILURE}
       [{+|-}{SUCCESS|FAILURE}"]
   [-server server_name] [-version] [-help]
```

Parameters

–names *db_names* – specifies the databases to modify, where *db_names* lists one or more existing database names. If the list of database names contains spaces, you must enclose the list in double quotation marks (see “Option Lists” on page 1-3). By default, **ssetdbaudit** modifies all databases in SQL Server. Collection path names are invalid here (use **–server** to specify a SQL Server collection object).

–wildcard – enables wildcard matching on *db_names* (see “Wildcards” on page 1-4).

–drop – configures auditing of successful and failed **drop** command usage.

{ **+** | **–** } **SUCCESS** | **FAILURE** – specifies auditing configuration for specified command. Use “+” to enable auditing and “–” to disable auditing; use SUCCESS to indicate auditing of successful usage and FAILURE to indicate auditing of permission failures.

Syntax	Meaning
+SUCCESS	Enable auditing of successful command usage
–SUCCESS	Disable auditing of successful command usage
+FAILURE	Enable auditing of failed command usage attempts
–FAILURE	Disable auditing of failed command usage attempts

–use – configures auditing of successful and failed **use** command usage.

–truncatetable – configures auditing of successful and failed **truncate table** command usage.

–grant – configures auditing of successful and failed **grant** command usage.

–revoke – configures auditing of successful and failed **revoke** command usage.

–access – configures auditing of successful and failed access from other databases. This audits execution of SQL commands from within another database that refer to objects in *db_names*.

–all – configures auditing of successful and failed usage of **all** database commands (**drop**, **use**, **truncate table**, **grant**, and **revoke**), along with successful and failed access from other databases.

–server *server_name* – specifies the SQL Server installation on which to set auditing information. The default is the current managed SQL Server collection (see “Command Context” on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
ssetdbaudit –drop –SUCCESS –grant +FAILURE –revoke
+FAILURE
```

Configures command auditing for all databases in the current SQL Server to exclude command auditing of successful uses of the **drop** command and to include auditing of permission failures in uses of the **grant** and **revoke** commands.

Comment

Before using **ssetdbaudit**, you must know the names of the databases to audit.

Permissions

To use **ssetdbaudit**, you must have the following roles:

TME	ESSM	SQL Server
user	security	sso_role

See Also

scrtauditrec, sgetdbaudit, sgetloginaudit, sgetprocaudit, sgetroleaudit, sgetserveraudit, sgettableaudit, sinstallaudit, ssetloginaudit, ssetprocaudit, ssetroleaudit, ssetserveraudit, ssettableaudit

ssetdefault

Function

Changes the properties of a default.

Syntax

```
ssetdefault -name default_name [-value expression]
   [ {-bind table_columns_or_datatypes |
      -unbind table_columns_or_datatypes}...[-future]]
   [-database db_path] [-version] [-help]
```

Parameters

–name *default_name* – specifies the name of the default to change. The name of the default can be prefixed with the name of the owner as follows: owner.default_name.

–value *expression* – specifies the new constant expression representing the default value for a column. Specifying this option causes the default to be deleted and recreated with the new value. The value must be surrounded by double quotes within single quotes.

–bind *table_columns_or_datatypes* – specifies a list of additional table columns, or user datatypes, or both to bind this default to. List members must be in the format *owner.datatype* or *owner.table.column*.

–unbind *table_columns_or_datatypes* – specifies a list of table columns, or user datatypes, or both from which to unbind this default. List members must be in the format *owner.datatype* or *owner.table.column*.

–future – prevents existing columns of the user datatype from acquiring (**–bind**) or losing (**–unbind**) the new default. Can be specified only with **–bind** or **–unbind**.

–database *db_path* – specifies the database in which the default exists. The default database is the current managed database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
ssetdefault -name sex_def -value '"M"'
-database PARIS/hrdb
```

This command changes the value of the default *sex_def*, located in the database *hrdb*, in the managed SQL Server PARIS, to the value “M”. Note that the value M is in two sets of quotes.

Permissions

To use **ssetdefault**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	Database Owner. If you are specifying bindings, you must be the owner of the object being bound to.

See Also

sgetdefault, scrtdefault, sdeldefault, scopydefault, scompdefault

ssetdev

Function

Modifies the configuration of a database device on a managed SQL Server.

Syntax

```
ssetdev –name device_name [–server server_name]
   [–default Boolean] [–enablemirror |
       [–disablemirror {PRIMARY | SECONDARY}]]
   [–version] [–help]
```

Parameters

–name *device_name* – specifies the logical name of the database device to modify, where *device_name* can be any existing device name in SQL Server.

–server *server_name* – specifies the managed SQL Server in which the device resides. The default is the current managed SQL Server collection (see “Command Context” on page 1-15).

–default *Boolean* – specifies whether to include the primary device in the managed SQL Server pool of default devices. TRUE includes the primary device in the managed SQL Server pool of default devices. FALSE removes the device from the pool of default space.

–enablemirror – enables the mirror for this device.

–disablemirror {PRIMARY | SECONDARY} – disables the primary device (PRIMARY) or the mirror (SECONDARY).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`ssetdev –n master –def FALSE`**

   Removes the *master* device from the pool of default space in the current managed SQL Server.

```
2. ssetdev –n dev1 –server BURL –disable SECONDARY
```

Temporarily disables, or suspends, the mirror device associated with the primary device *dev1* in the managed SQL Server BURL.

```
3. ssetdev –n masterdev –server BOSTON
       –disable PRIMARY
```

Temporarily disables, or suspends, the operation of the primary device *masterdev* in the managed SQL Server BOSTON. The secondary device remains active.

Comments

- Before using **ssetdev**, you need the name of:
  - The device to modify
  - The managed SQL Server containing the device
- See **scrtmirror** to create the secondary device
- See **sdeldev** to drop the primary or secondary device (or both)

Permissions

To use **ssetdev**, you must have the following roles:

TME	ESSM	SQL Server
user	space	sa_role

See Also

scrtdev, srecrtdev, scompdev, scopydev, sdeldev, sgetdev, scrtmirror

ssetgroup

Function

Modifies a group's membership.

Syntax

```
ssetgroup –name group_name [–addusers user_names]
   [–dropusers user_names] [–database db_path]
   [–version] [–help]
```

Parameters

–name *group_name* – specifies the name of the group to modify, where *group_name* can be any existing group name in the database.

–database *db_path* – specifies the database in which *group_name* exists, where *db_path* is a valid managed database name (see "Database Names" on page 1-16). The default is the current database collection (see "Command Context" on page 1-15).

–addusers *user_names* – specifies the name or names of the users to add as members of the group. Existing members of the group are unchanged.

–dropusers *user_names* – specifies the name or names of the users to remove from the group. Other members of the group are unchanged.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
ssetgroup –name nkotb –adduser allison
```

Changes the user membership of the group named *nkotb* in the current managed database to include user *allison*.

Comments

- Before using **ssetgroup**, you need the name of the database to access.
- The user names and group name must exist in the database.

Permissions

To use **ssetgroup**, you must have the following roles:

TME	ESSM	SQL Server
user	**security**	**sa_role** or Database Owner

See Also

scrtgroup, scompgroup, scopygroup, sdelgroup, sgetgroup

ssetindex

Function

Changes the properties of an index.

Syntax

```
ssetindex -name index_name [-columns column_names] [-
   segment segment_name]  [-unique | -nonunique] [-
   fillfactor percentage]
   [-clustered | -nonclustered]
   [-ignoredupkey | noignoredupkey]
   [-duprow { ’allow’ | ’disallow’ | ’ignore’ }]
   [-updatestats]  [-database db_path] [-version] [-
   help]
```

Parameters

–name *index_name* – specifies the name of the index to change in the format owner.table.indexname.

–columns *column_names* – specifies the names of the new columns to set the index on. These columns, if specified, replace all previous columns representing this index. If you do not use **–columns**, the columns comprising the index are not modified.

–segment *segment_name* – specifies the segment to move the index to. If the index has to be recreated (because additional switches are being supplied to this command), it is moved to the specified segment. If this is the only option specified, Enterprise SQL Server Manager modifies the segment, so that all future allocations will occur on the specified segment, rather than deleting and recreating the index, a time consuming task.

–unique – prohibits duplicate index (key) values. The default for uniqueness is **–nonunique**.

–nonunique – allows duplicate index (key) values. This is the default for uniqueness.

–fillfactor – specifies how full SQL Server will make each page when it creates a new index on existing data. Valid values are 1 through 100. The default value is 0.

–clustered – makes the physical order of the rows on this database device the same as the indexed order of the rows. The default value for clustering is **–nonclustered**.

–nonclustered – specifies that there is a level of indirection between the index structure and the data itself. This is the default value for clustering.

–ignoredupkey – tells SQL Server to ignore attempts to enter duplicate keys into the table.

–noignoredupkey – tells SQL Server to disallow attempts to enter duplicate keys into the table.

–duprow {"allow" | "disallow" | "ignore"} – tells SQL Server how to handle duplicate rows during index creation. If you specify **"allow"**, the server allows duplicate rows to exist in the table. If you specify **"disallow"**, the index creation fails if duplicate rows are encountered. If you specify **"ignore"**, SQL Server ignores duplicate rows (essentially, gets rid of them) when creating the index. This option is not relevant when creating a non-unique, nonclustered index. The default is **"disallow"**.

–updatestats – updates information about the distribution of key values in the specified index. Changing the attributes of an index causes statistics to be updated automatically. If you change the index and specify **–updatestats**, Enterprise SQL Server Manager updates statistics only one time, thus eliminating redundant processing.

–database *db_path* – specifies the database in which the table exists. The default is the current managed database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Changing the attributes of an index causes the index to be deleted and recreated with the new values. This can be a time-consuming operation.

Example

```
ssetindex -table salary_table -name primary_key
-clustered -database PARIS/hrdb
```

This command changes the index *primary_index* to a clustered index. This index is located in the table *salary_table*, in the database *hrdb*, in the managed SQL Server PARIS.

Permissions

To use **ssetindex**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	table and index owner

See Also

scheckindex, **sgetindex**, **scrtindex**, **sdelindex**, **scopyindex**, **scompindex**

ssetlogin

Function

Modifies the configuration of a login account in SQL Server.

Syntax

```
ssetlogin –name login_name [–fullname full_name]
   [–defaultdatabase db_name]
   [–language default_language] [–lock Boolean]
   [[–password new_password] |
       [-sybasepassword new_password]]
   [–addroles roles] [–droproles roles]
   [–server server_name] [–version] [–help]
```

Parameters

–name *login_name* – specifies the name of the user login to modify.

–fullname *full_name* – specifies the full name that corresponds to the user login. If the full name includes spaces, enclose the entire name with quotation marks: “John Doe”.

–defaultdatabase *db_name* – specifies the default database.

–language *default_language* – specifies the default language. Permissible languages include: **english**, **french**, etc. You can list the available languages using **sgetserver**.

–lock *Boolean* – specifies whether to lock the account (TRUE to lock; FALSE to unlock). This option accepts any Boolean value (see “Boolean Keywords” on page 1-3).

–password *new_password* – specifies the new password for the login account.

–sybasepassword *new_password* – sets a SQL Server login account for this login in a specific managed SQL Server. Using this option is equivalent to executing the **ssetsybaselogin** command:

```
ssetsybaselogin –principal name –password password
–server server
```

–addroles *roles* – adds the specified roles to those that this login already assumes. Permissible values for role are **sa_role**, **sso_role**, and **oper_role** and must be lowercase. When specifying multiple roles, separate the roles by a space and enclose the string with quotation marks: “sa_role sso_role”.

–droproles *roles* – removes the specified roles from those that this login already assumes. Permissible values for role are **sa_role**, **sso_role**, and **oper_role** and must be lowercase. When specifying multiple roles, separate the roles by a space and enclose the string with quotation marks: “sa_role sso_role”.

–server *server_name* – specifies the managed SQL Server in which to change the account. The default is the current managed SQL Server collection (see “Command Context” on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`ssetlogin –n jojo –droproles sa_role`**

   Removes System Administrator privileges (**sa_role**) from the user login *jojo*.

2. **`ssetlogin –n bwebster –F "Bob Webster" –da pubs2`**

   Modifies the user login *bwebster* to change full name to “Bob Webster,” and change the default database to *pubs2*.

3. **`ssetlogin –n liz –p newpwd`**

   Modifies the user login *liz* to change the password to *newpwd*.

4. **`ssetlogin –n joe –addroles sa_role –drop "sso_role"`**

   Modifies the user login *joe* to add the System Administrator role (**sa_role**) and remove the System Security Officer role (**sso_role**).

Comments

- Before using **ssetlogin**, you need the following information:
  - The name of a particular login to change
  - The SQL Server installation containing the login
  - The details of the login attributes being changed
  - It is helpful to have knowledge of the existing roles a login has prior to updating the role list (see **sgetlogin**).

- This command removes all password information from the command line to prevent unauthorized users from viewing the password via the UNIX **ps** command.
- If **ssetlogin** fails, the command does not roll back modifications to the login's password and the login retains the new password.

Permissions

To use **ssetlogin**, you must have the following roles:

TME	ESSM	SQL Server
any. To use the **–sybasepassword** option, you must have **super** role at the TMR level.	**security**	**sso_role** and **sa_role**

See Also

scrtlogin, scrtsybaselogin, scomplogin, scopylogin, sdellogin, sgetlogin, ssetloginaudit, ssetsybaselogin, ssetuser

ssetloginaudit

Function

Configures auditing of one or more login accounts in SQL Server, in order to audit those logins' attempts to access tables and views, or to audit the text of the logins' command batches.

Syntax

```
ssetloginaudit [–names login_names [–wildcard]]
   [–batch Boolean]
   [–table "{+|–}{SUCCESS|FAILURE}
       [{+|–}{SUCCESS|FAILURE}]"]
   [–view "{+|–}{SUCCESS|FAILURE}
       [{+|–}{SUCCESS|FAILURE}]"]
   [–all "{+|–}{SUCCESS|FAILURE}
       [{+|–}{SUCCESS|FAILURE}]"]
   [–server server_name] [–version] [–help]
```

Parameters

–names *login_names* – specifies the name or names of the user logins to audit. If you omit this option, **ssetloginaudit** sets up auditing for all logins associated with SQL Server. When specifying multiple names, separate the names with spaces and enclose the string with quotation marks (see "Option Lists" on page 1-3).

–wildcard – enables wildcard matching on *login_names* (see "Wildcards" on page 1-4).

–batch *Boolean* – TRUE enables batch execution auditing. If enabled, SQL Server audits all SQL statements executed by the logins. FALSE disables batch auditing.

–table – configures auditing of successful and failed table access.

{ **+** | **–** } **SUCCESS** | **FAILURE** – specifies auditing configuration for specified access type. Use "+" to enable auditing and "–" to disable auditing; use SUCCESS to indicate auditing of successful access attempts and FAILURE to indicate auditing of failed access attempts.

Syntax	Meaning
+SUCCESS	Enable auditing of successful access
–SUCCESS	Disable auditing of successful access attempts
+FAILURE	Enable auditing of failed access attempts
–FAILURE	Disable auditing of failed access attempts

–view – configures auditing of successful and failed view accesses.

–all – configures auditing of successful and failed table access **and** view access.

–server *server_name* – specifies the SQL Server installation in which to audit the logins. The default is the current managed SQL Server collection (see "Command Context" on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`ssetloginaudit –n jojo –b TRUE`**

   Enables batch execution auditing for user login *jojo* in the current SQL Server. Any other auditing options that might have been set previously for login *jojo* are not changed.

2. **`ssetloginaudit –n "barneyb harryc" –t +FAILURE –s BOSTON`**

   Enables auditing of table access permission failures for user logins *barneyb* and *harryc* in SQL Server BOSTON.

3. **`ssetloginaudit –name mj –view "+SUCCESS –FAILURE"`**

   Enables auditing of successful view accesses and disables auditing of view access permission failures for user login *mj* in the current SQL Server.

4. **`ssetloginaudit –all "+SUCCESS +FAILURE"`**

   Enables auditing of successful and failed table and view accesses for all logins in the current SQL Server.

Comments

- Before using **ssetloginaudit**, you need the following information:
  - The login names to be audited
  - The name of the SQL Server installation containing the logins
- See **sgetroleaudit** and **ssetroleaudit** to get and set auditing information for any roles (**sa_role**, **sso_role**, **oper_role**) that may be associated with a login.

Permissions

To use **ssetloginaudit**, you must have the following roles:

TME	ESSM	SQL Server
any	**security**	**sso_role**

See Also

scrtauditrec, sgetdbaudit, sgetloginaudit, sgetprocaudit, sgetroleaudit, sgetserveraudit, sgettableaudit, sinstallaudit, ssetdbaudit, ssetprocaudit, ssetroleaudit, ssetserveraudit, ssettableaudit

ssetobjperm

Function

Grants or revokes permission to use the following actions on specified tables, views, and stored procedures:

- **select**
- **update**
- **insert**
- **delete**
- **references**
- **execute**

Syntax

```
ssetobjperm -name object
   {-grant | -revoke [-cascade]}
   [[-actions action_names] | –all]
   {[-users user_names [-with_grant_option]]
       | [-groups group_names]
       | [-roles role_names]}...}
   [-database db_path] [-version] [-help]
```

Parameters

–name *object* – specifies the object to which the permission applies.

–grant – grants permission to specified users, groups, or roles.

–revoke – revokes permission from specified users, groups, or roles.

–cascade – revokes permission from all who were granted permission by the **–with_grant_option.**

–actions *action_names* – specifies the actions that can be performed on the specified object.

–all – applies grant or revoke permissions for all actions.

–users *user_names* – specifies the users for which you are configuring object permissions. If the list contains spaces, enclose the whole list in double quotation marks (see “Option Lists” on page 1-3).

–with_grant_option – gives users specified in *user_names* permission to grant the permission to others. You cannot grant permissions with the grant option to a group or a role.

–groups *group_names* – specifies the groups for which you are configuring object permissions. If the list contains spaces, enclose the whole list in double quotation marks (see “Option Lists” on page 1-3).

–roles *role_names* – specifies the roles for which you are configuring object permissions. If the list contains spaces, enclose the whole list in double quotation marks (see “Option Lists” on page 1-3).

–database *db_path* – specifies the database on which to get permission information. The default is the current database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
ssetobjperm -name testTable -actions update
-data AGRA/pubs2 -revoke -users bwebster
```

Revokes **update** permissions on the *testTable* table from user *bwebster* in the *pubs2* database.

Comments

- Before using **ssetobjperm**, you need the name of the database to access.
- If you do not specify **–actions** or **–al**l, the default is to set permissions on all actions.

Permissions

To use **ssetobjperm**, you must have the following roles:

TME	ESSM	SQL Server
any	**security**	**sa_role** or Database Owner

See Also

sgetcmdperm, sgetobjperm, ssetobjperm

ssetprocaudit

Function

Audits the execution of stored procedures and triggers in a database.

Syntax

```
ssetsprocaudit [–names proc_names [–wildcard]] [–new]
   [–execution "{+|–}{SUCCESS|FAILURE}
       [{+|–}{SUCCESS|FAILURE}]"]
   [–database db_path] [–version] [–help]
```

Parameters

–names *proc_names* – specifies the existing stored procedures and triggers to audit. If you omit this option, **ssetsprocaudit** sets up auditing for all existing procedures and triggers in a database. Separate multiple names with a space and enclose the entire string with quotation marks: “proc1 proc2 trigger1”.

–wildcard – enables wildcard matching on *sprocs* (see “Wildcards” on page 1-4).

–new – specifies that all new stored procedures and triggers are to be audited as specified.

–execution – configures auditing of successful and failed execution for the specified stored procedures and triggers (those listed in *proc_names,* or all new stored procedures and triggers, or both).

{ **+** | **–** } **SUCCESS** | **FAILURE** – specifies auditing configuration for specified stored procedures and triggers. Use “+” to enable auditing and “–” to disable auditing; use SUCCESS to indicate auditing of successful usage and FAILURE to indicate auditing of permission failures.

Syntax	Meaning
+SUCCESS	Enable auditing of successful stored procedure and trigger execution attempts
–SUCCESS	Disable auditing of successful stored procedure and trigger execution attempts
+FAILURE	Enable auditing of failed stored procedure and trigger execution attempts
–FAILURE	Disable auditing of failed stored procedure and trigger execution attempts

–database *db_path* – specifies the database for which to set auditing information. The default is the current managed database resource collection (see "Command Context" on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
ssetsprocaudit –n add_title –E "+SUCCESS +FAILURE"
–d pubs2
```

Configures stored procedure auditing of both successful and failed executions of the *add_title* stored procedure in the *pubs2* database.

Comments

Before using **ssetprocaudit**, you need the following information:

- The stored procedures and triggers to audit.
- The name of the database containing the procedures and triggers

Permissions

To use **ssetprocaudit**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	**sso_role**

See Also

scrtauditrec, sgetdbaudit, sgetloginaudit, sgetprocaudit, sgetroleaudit, sgetserveraudit, sgettableaudit, sinstallaudit, ssetdbaudit, ssetloginaudit, ssetroleaudit, ssetserveraudit, ssettableaudit

ssetrmtlogin

Function

Modifies remote login configuration options.

Syntax

```
ssetrmtlogin –names remote_login_names [–wildcard]
   –remoteserver remote_server_name
   [–locallogin local_login_name] [–trusted Boolean]
   [–server server_name] [–version] [–help]
```

Parameters

–names *remote_login_names* – specifies the names of the remote logins to configure.

–wildcard – enables wildcard matching on *remote_login_names* (see “Wildcards” on page 1-4).

–remoteserver *remote_server_name* – specifies the name of the remote SQL Server.

–locallogin *local_login_name* – explicitly maps the remote login to the specified local login name.

–trusted *Boolean* – FALSE specifies that SQL Server should perform password checking for the remote login (untrusted mode); TRUE specifies that SQL Server should not (trusted mode). This option accepts any Boolean argument value (see “Boolean Keywords” on page 1-3).

–server *server_name* – specifies the local managed SQL Server. The default is the current managed SQL Server collection (see “Command Context” on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
ssetrmtlogin –remoteserver SPAIN –n jose –trusted
FALSE
```

Modifies the trusted mode configuration for the remote login named *jose* so that the password is verified.

Comments

Before using **ssetrmtlogin**, you need the following information:

- The names of the remote logins to configure
- The name of the remote SQL Server
- The local managed SQL Server

Permissions

To use **ssetrmtlogin**, you must have the following roles:

TME	ESSM	SQL Server
user	security	sa_role

See Also

scrtrmtlogin, sdelrmtlogin, sgetremotelogin, scrtrmtserver, sdelrmtserver, sgetrmtserver, ssetrmtserver, sconfigserver

ssetrmtserver

Function

Modifies the remote SQL Server configuration for a specified local SQL Server and remote SQL Server pair.

Syntax

```
ssetrmtserver –name remote_server_name
   [–server server_name] [–timeout Boolean]
   [–encrypt Boolean]
   [[–locallogin login_name] | –remotelogin
       | –nologinmap]
   [–version] [–help]
```

Parameters

–name *remote_server_name* – specifies the name of the remote SQL Server for which to modify remote SQL Server options.

–server *server_name* – specifies the name of the local managed SQL Server in which the remote SQL Server is configured. The default is the current managed SQL Server collection (see “Command Context” on page 1-15).

–timeout *Boolean* – enables or disables the timeout capability used by the local SQL Server. Choose TRUE to drop the physical network connection to the remote SQL Server after one minute of inactivity. Choose FALSE so that the connection will not automatically be dropped. This option accepts any Boolean argument value (see “Boolean Keywords” on page 1-3).

–encrypt *Boolean* – enables or disables encryption of passwords for network transmission. Choose TRUE for encrypted passwords; FALSE for unencrypted passwords. This option accepts any Boolean argument value (see “Boolean Keywords” on page 1-3).

–locallogin *login_name* – option to change the default login name strategy. This option maps all logins from the remote SQL Server to a single local login name.

–remotelogin – option to change the default login name strategy. This option allows all logins from the remote SQL Server to use their own names as local login names.

–nologinmap – option to change the default login name strategy. This option removes the current default login mapping. Without a default mapping, remote logins must be mapped individually to particular local login names.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
ssetrmtserver –name EMERY –S BOSTON –remote
```

Changes the local login mapping, so that all logins from the remote SQL Server EMERY can use their remote names on the local managed SQL Server BOSTON. You may override this login strategy by changing the configuration for an individual remote login. (See **scrtrmtlogin** and **ssetrmtlogin**.)

Comments

- Before using **ssetrmtserver**, you need the name of:
  - The remote SQL Server to configure
  - The local managed SQL Server
- You can configure remote access options for the local managed SQL Server (such as enabling and disabling remote access and specifying maximum number logins, maximum SQL Servers, and maximum connections) by using the **sconfigserver** command.
- You can configure remote login options by using the **ssetrmtlogin** command.

Permissions

To use **ssetrmtserver**, you must have the following roles:

TME	ESSM	SQL Server
user	security	sa_role

See Also

scrtrmtserver, sdelrmtserver, sgetrmtserver, scrtrmtlogin, sdelrmtlogin, sgetremotelogin, ssetrmtlogin, sconfigserver

ssetroleaudit

Function

Configures auditing the use of privileged commands for roles in a specified SQL Server. Role types are **sa_role**, **sso_role**, **oper_role**.

Syntax

```
ssetroleaudit [–types roles]
   [–command "{+|–}{SUCCESS|FAILURE}
       [{+|–}{SUCCESS|FAILURE}]"]
   [–server server_name] [–version] [–help]
```

Parameters

–types *roles* – specifies the roles to audit. The list of roles is: **sa_role**, **sso_role**, and **oper_role**. If you omit this option, **ssetroleaudit** sets up auditing for all roles associated with a SQL Server.

–command – configures auditing of successful and failed privileged command usage. If you do not specify **–command**, **ssetroleaudit** does not configure auditing.

{ **+** | **–** } **SUCCESS** | **FAILURE** – specifies auditing configuration for privileged command usage. Use "+" to enable auditing and "–" to disable auditing; use SUCCESS to indicate auditing of successful usage and FAILURE to indicate auditing of permission failures.

Syntax	Meaning
+SUCCESS	Enable auditing of successful command usage
–SUCCESS	Disable auditing of successful command usage
+FAILURE	Enable auditing of failed command usage attempts
–FAILURE	Disable auditing of failed command usage attempts

–server *server_name* – specifies the SQL Server installation in which to configure auditing. The default is the current managed SQL Server collection (see “Command Context” on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`ssetroleaudit –types "sa_role sso_role" –command +FAILURE –server BURL`**

   Configures auditing in SQL Server BURL to include command auditing of privilege violations for the **sa_role** and **sso_role**.

2. **`ssetroleaudit –c "–SUCCESS +FAILURE" –s BURL`**

   Configures auditing of all roles in SQL Server BURL to disable auditing of successful command uses and to include command auditing of privilege violations.

Comments

Before using **ssetroleaudit**, you need the following information:

- The role types to configure for auditing
- The name of the SQL Server installation

Permissions

To use **ssetroleaudit**, you must have the following roles:

TME	ESSM	SQL Server
any	**security**	**sso_role**

See Also

scrtauditrec, sgetdbaudit, sgetloginaudit, sgetprocaudit, sgetroleaudit, sgetserveraudit, sgettableaudit, sinstallaudit, ssetdbaudit, ssetloginaudit, ssetprocaudit, ssetserveraudit, ssettableaudit

ssetrule

Function

Changes the properties of a rule.

Syntax

```
ssetrule -name rule_name
   {[-bind table_columns_or_datatypes]
       [-unbind table_columns_or_datatypes] [-future]}
   [-database db_path][-version] [-help]
```

Parameters

–name *rule_name* – specifies the name of the rule to change. The name of the rule can be prefixed with the name of the owner as follows: *owner.rule_nam*e.

–bind *table_columns_or_datatypes* – specifies a list of additional table columns, or user datatypes, or both to bind this rule to. List members must be in the format *owner.datatype* or *owner.table.column*.

–unbind *table_columns_or_datatypes* – specifies a list of table columns, or user datatypes, or both from which to unbind this rule. List members must be in the format *owner.datatype* or *owner.table.column*.

–future – prevents existing columns of the user datatype from acquiring (**–bind**) or losing (**–unbind**) the new rule. Can be specified only with **–bind** or **–unbind**.

–database *db_path* – specifies the database in which the rule exists. The default is the current managed database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
ssetrule -name dbo.id_rule -database PARIS/master -
bind id_type
```

This command changes the rule *id_rule*, located in the database *master*, in the managed SQL Server PARIS by binding it to the datatype *id_type*.

Permissions

To use **ssetrule**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	Database Owner. If you are specifying bindings, you must be the owner of the object being bound to.

See Also

sgetrule, scrtrule, sdelrule, scopyrule, scomprule

ssetseg

Function

Modifies the attributes of a segment. Extends a segment to span additional existing database devices, or removes devices from a segment.

Syntax

```
ssetseg –name segment_name [–adddevices device_names]
   [–dropdevices device_names] [–database db_path]
   [–version] [–help]
```

Parameters

–name *segment_name* – specifies the name of the segment to modify.

–adddevices *device_names* – specifies the logical names of the devices on which to extend a segment. Separate device names with a space and enclose the list with double quotation marks.

–dropdevices *device_names* – reduces the expanse of the segment by removing the specified logical devices from the segment. Separate device names with a space and enclose the list with double quotation marks.

–database *db_path* – specifies the database in which the segment is defined. If you omit this option, **ssetseg** applies to segments in the current managed database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. `ssetseg –n seg1 –add dev2 –database BOSTON/pubs2`

   Extends a segment named *seg1* onto device *dev2* in the managed SQL Server BOSTON.

2. `ssetseg –name myseg –drop dev2 –add dev1 –database BOSTON/pubs2`

   Removes the device *dev2* from the segment named *myseg*, and adds the device *dev1*.

Comments

- Before using **ssetseg**, you need the following information:
  - The database on which you wish to extend a segment.
  - The name of the segment to modify.
  - The names of the devices to extend or remove a segment from.
- When adding a device, the device must exist and be available to the database. You can create the device with **scrtdev**, and make it available using **scrtdb** or **ssetdb**.
- When removing a device, the segment must name at least one other device.
- The segment must exist in the database. You can create the segment with **scrtseg**.

Permissions

To use **ssetseg**, you must have the following roles:

TME	ESSM	SQL Server
any	**space**	**sa_role**

See Also

scrtdev, sgetdev, scrtseg, scompseg, scopyseg, sdelseg, sgetseg, scrtdb, ssetdb

ssetserver

Function

Modifies SQL Server configuration variables and executes a SQL Server **reconfigure** command to cause dynamic variables to take effect immediately.

Syntax

```
ssetserver [–name server_name]
   [–config "'config_name' config_value
       [, 'config_name' config_value]..."]
   [-errorlog filename] [–override]
   [–version] [–help]
```

Parameters

–name *server_name* – specifies the SQL Server installation to modify, where *server_name* can be any valid managed SQL Server name (see “SQL Server Names” on page 1-15). The default is the current managed SQL Server collection (see “Command Context” on page 1-15).

–config – specifies one or more configuration variables that you want to modify. See Appendix A, "SQL Server Configuration Parameters” in *Enterprise SQL Server Manager User’s Guide* for additional information about configuration variables.

config_name – specifies the name of a configuration variable to modify.

config_value – specifies the new value you want given to the variable specified by *config_name*.

–override – reconfigures with override. You must include the **–override** option if you want to modify the **allow updates** configuration variable. The default is to reconfigure without override.

–errorlog ***filename*** – specifies the full pathname of the SQL Server error log. Event Monitoring Services use this information to monitor the error log. The error log must be on the management host.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
ssetserver –name BOSTON –config "'recovery
interval' 3, password% 0"
```

Sets the **recovery interval** in SQL Server BOSTON to 3 minutes and disables the **password** expiration policy.

Comments

- Before using **ssetserver**, you need the following information:
  - The name of the SQL Server installation you want to configure
  - The configuration variables that you want to modify
- By default, wildcard use is enabled within **–config** parameter entries.
- See “Option Lists” on page 1-3 for additional information about argument lists.
- To modify remote Server attributes, see the **ssetrmtserver** command.
- In addition to having the role, **sa_role**, you must have **sso_role** to set the following SQL Server configuration variables:
  - **allow updates**
  - **audit queue size**
  - **password expiration interval**
  - **remote access**

Permissions

To use **ssetserver**, you must have the following roles:

TME	ESSM	SQL Server
any	**server**	**sa_role**

See Also

smanageserver, sunmanageserver, scompserver, scopyserver, sgetserver, sgetprocess, skillprocess, sstartserver, sstopserver, ssetrmtserver

ssetserveraudit

Function

Configures system–wide auditing and auditing of SQL Server events iowner.rule_namn a specified SQL Server.

Syntax

```
ssetserveraudit [–name server_name]
   [–audit {ON|OFF}]
   [–logins "{+|–}{SUCCESS|FAILURE}
       [{+|–}{SUCCESS|FAILURE}]"]
   [–logouts {ON|OFF}]
   [–connections "{+|–}{SUCCESS|FAILURE}
       [{+|–}{SUCCESS|FAILURE}]"]
   [–setroles "{+|–}{SUCCESS|FAILURE}
       [{+|–}{SUCCESS|FAILURE}]"]
   [–errors "{+|–}{FATAL|NONFATAL}
       [{+|–}{FATAL|NONFATAL}]"]
   [–startups {ON|OFF}] [–adhoc {ON|OFF}]
   [–version] [–help]
```

Parameters

–name *server_name* – specifies the SQL Server installation for which to set auditing information. The default is the current managed SQL Server collection (see “Command Context” on page 1-15).

–audit { **ON** | **OFF** } – enables (ON) or disables (OFF) system–wide auditing in the specified SQL Server. This option must be enabled before setting any of the other auditing options. If you turn auditing off, SQL Server retains the current auditing configuration settings, so that when you later re–enable auditing, you can then continue to log those previously configured auditing events.

–logins – configures auditing of successful and failed login attempts.

{ **+** | **–** }{ **SUCCESS** | **FAILURE** } – specifies auditing configuration for SQL Server event. Use “+” to enable auditing and “-” to disable auditing; use SUCCESS to indicate auditing of successful usage and FAILURE to indicate auditing of permission failures.

Syntax	Meaning
+SUCCESS	Enable auditing of successful usage
–SUCCESS	Disable auditing of successful usage
+FAILURE	Enable auditing of failed usage attempts
–FAILURE	Disable auditing of failed usage attempts

–logouts { **ON** | **OFF** } – turns auditing of logouts ON or OFF.

–connections – configures auditing of successful and failed RPC connection attempts.

–setroles – configures auditing of successful and failed attempts to set roles (**sa_role**, **sso_role**, **oper_role**).

–errors – configures auditing of fatal / nonfatal user errors.

{ **+** | **–** }{ **FATAL** | **NONFATAL** } – specifies auditing configuration for user errors. Use "+" to enable auditing and "–" to disable auditing; use FATAL to indicate auditing of fatal user errors and NONFATAL to indicate auditing of nonfatal user errors.

Syntax	Meaning
+FATAL	Enable auditing of fatal user errors
–FATAL	Disable auditing of fatal user errors
+NONFATAL	Enable auditing of nonfatal user errors
–NONFATAL	Disable auditing of nonfatal user errors

–startups { **ON** | **OFF** } – turns auditing of SQL Server startups ON or OFF.

–adhoc { **ON** | **OFF** } – turns auditing of ad hoc additions to audit trail ON or OFF.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`ssetserveraudit –logins "+SUCCESS +FAILURE" –server BURL`**

   Configures SQL Server auditing in managed SQL Server BURL to include event auditing of both successful and failed login attempts.

2. **`ssetserveraudit –audit OFF –server BURL`**

   Disables all auditing in managed SQL Server BURL until auditing is re–enabled. Enterprise SQL Server Manager retains the status of the individual auditing options.

Comment

Before using **ssetserveraudit**, you need the name of the SQL Server installation to audit.

Permissions

To use **ssetserveraudit**, you must have the following roles:

TME	ESSM	SQL Server
any	**security**	**sso_role**

See Also

scrtauditrec, sgetdbaudit, sgetloginaudit, sgetprocaudit, sgetroleaudit, sgetserveraudit, sgettableaudit, sinstallaudit, ssetdbaudit, ssetloginaudit, ssetprocaudit, ssetroleaudit, ssettableaudit

ssetsybaselogin

Function

Defines a SQL Server login for an administrator. A login can be defined for the Tivoli management region (TMR), for a policy region, or for a particular managed SQL Server.

Syntax

```
ssetsybaselogin –principal username
   –password password [–name server_login_name]
   [[–policyregion policyregion] | [–server server]]
   [-version] [-help]
```

Parameters

–principal *username* – specifies the Tivoli name for the user. On UNIX systems, this is the UNIX login name.

–password *password* – specifies the SQL Server password for the principal *username*.

–name *server_login_name* – specifies the SQL Server login name. If not specified, the principal *username*.

–policyregion *policyregion* – specifies the specific policy region for which the password and SQL Server login name are set.

–server *server* – specifies the managed SQL Server for which the password and SQL Server login name are set.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
ssetsybaselogin -principal joe -password july4
–name joed -server KOKO
```

Sets the SQL Server login for TME user *joe,* with the password *july4,* in the managed Server KOKO. The administrator has the SQL Server login *joed.*

Comments

- If an entry for an administrator does not exist, **ssetsybaselogin** creates one. If an entry does exist, **ssetsybaselogin** updates the password.
- If you omit both the **–policyregion** and **–server** options, the SQL Server login is set for the Tivoli Management Region.
- If you create a subregion in a policy region, an administrator must have a SQL Server login for that subregion in order to access it. The SQL Server login for the parent policy region will not enable access to a subregion.

Permissions

To use **ssetsybaselogin**, you must have the following roles:

TME	ESSM	SQL Server
super	none	none

See Also

scrtlogin, scrtsybaselogin, sdelsybaselogin, ssetlogin

ssettable

Function

Changes the properties of a table.

Syntax

```
ssettable -name table_name [-segment segment_name]
   [-updatestats] [-truncate] [-recompile]
   [-constraint "’constraint_name’ ’constraint_text’,
   ’constraint_name’ ’constraint_text’, ...”]
   [-drop constraint_name] [-database db_path] [-
   version] [-help]
```

Parameters

–name *table_name* – specifies the name of the table to change. The name of the table can be prefixed with the name of the owner as follows: owner.table_name.

–segment *segment_name* – specifies the name of the segment on which to place future space allocation for the table.

–updatestats – updates information about the distribution of key values in all indexes in the table.

–truncate – deletes all rows from the table.

–recompile – causes each stored procedure and trigger that uses the table to be recompiled the next time they are executed. Specifying this option after changing a table's indexes or changing a database in ways that affect its statistics can improve the operating efficiency of stored procedures and triggers.

–constraint *‘constraint_name’ ‘constraint_text’* – specifies a list of constraints to be set on a table. This option sets the following table level constraints:

- referential integrity
- check
- unique
- primary key

You can use **-constraint** to set one or more constraints of any type. The constraint name field is the name of the constraint and the text is the full text of the constraint command.

–drop *constraint_name* – drops the specified constraint.

–database *db_path* – specifies the database in which the table exists. The default is the current managed database.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
ssettable -name salary_table -updatestats
-database HR1/FY95
```

This command changes the table *salary_table*, located in the database *FY95*, in the managed SQL Server HR1 by updating information about the distribution of key values in all indexes in the table.

Examples of constraint syntax

To create a referential integrity constraint:

```
-constraint "'my_constraint' 'foreign key (stor_id, ord_num)
references my_sales (stor_id, ord_num)'"
```

You can specify the table name following the reference's key word with the syntax *database.owner.tablename*.

To create a check constraint:

```
-constraint "'my_check_constraint' 'check(pub_id in ("1389",
"0736", "0877") or pub_name not like "Bad News Books")'"
```

To create a unique / primary key constraint:

```
-constraint "'my_primary_constraint' 'unique clustered
(stor_id, ord_num)'"
```

Permissions

To use **ssettable**, you must have the following roles:

TME	ESSM	SQL Server
any	**schema**	For the **-truncate** and **-updatestats** options, you must be table owner. For the **-segment** option, you must be either table owner, Database Owner, or have **sa_role**. For all other options you must be Database Owner.

See Also

sgettable, scrttable, sdeltable, scopytable, scomptable, schecktable

ssettableaudit

Function

Configures auditing of accesses to tables and views in a database.

Syntax

```
ssettableaudit [–names tables_and_views [–wildcard]]
   [–new]
   [–delete "{+|–}{SUCCESS|FAILURE}
       [{+|–}{SUCCESS|FAILURE}"]
   [–insert "{+|–}{SUCCESS|FAILURE}
       [{+|–}{SUCCESS|FAILURE}"]
   [–select "{+|–}{SUCCESS|FAILURE}
       [{+|–}{SUCCESS|FAILURE}"]
   [–update "{+|–}{SUCCESS|FAILURE}
       [{+|–}{SUCCESS|FAILURE}"]
   [–all "{+|–}{SUCCESS|FAILURE}
       [{+|–}{SUCCESS|FAILURE}"]
   [–database db_path] [–version] [–help]
```

Parameters

–names *tables_and_views* – specifies the existing tables and views to audit. Without this option, **ssettableaudit** enables auditing for all existing tables and views in a database.

–wildcard – enables wildcard matching on *tables_and_views* (see “Wildcards” on page 1-4).

–new – specifies that the audit settings are to be the defaults for newly created tables and views in the database.

–delete – configures auditing of successful and failed **delete** command usage in the specified tables and views.

{ **+** | **–** } **SUCCESS** | **FAILURE** – specifies auditing configuration for specified command. Use “+” to enable auditing and “–” to disable auditing; use SUCCESS to indicate auditing of successful usage and FAILURE to indicate auditing of permission failures.

Syntax	Meaning
+SUCCESS	Enable auditing of successful command usage
–SUCCESS	Disable auditing of successful command usage
+FAILURE	Enable auditing of failed command usage attempts
–FAILURE	Disable auditing of failed command usage attempts

–insert – configures auditing of successful and failed **insert** command usage in the specified tables and views.

–select – configures auditing of successful and failed **select** command usage in the specified tables and views.

–update – configures auditing of successful and failed **update** command usage in the specified tables and views.

–all – configures auditing of successful and failed usage for **all** table commands.

–database *db_path* – specifies the database for which to set auditing information. The default is the current database collection (see “Command Context” on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
ssettableaudit –names "authors titles" –all
"–SUCCESS +FAILURE" –select –SUCCESS –database
pubs2
```

Configures auditing of successful and failed executions of all commands except successful executions of **select** in the *authors* and *titles* tables in the *pubs2* database.

Comments

Before using **ssettableaudit**, you need the following information:

- The tables and views to audit
- The database of the tables and views to audit

Permissions

To use **ssettableaudit**, you must have the following roles:

TME	ESSM	SQL Server
any	**security**	**sso_role**

See Also

scrtauditrec, sgetdbaudit, sgetloginaudit, sgetprocaudit, sgetroleaudit, sgetserveraudit, sgettableaudit, sinstallaudit, ssetdbaudit, ssetloginaudit, ssetprocaudit, ssetroleaudit, ssetserveraudit

ssetthresh

Function

Modifies the definition of a threshold on a segment in a database.

Syntax

```
ssetthresh –name "segment_name free_pages"
   [–freespace num_pages] [–sproc proc_name]
   [–database db_path] [–version] [–help]
```

Parameters

–name *segment_name free_pages* – specifies the threshold to modify. The threshold is identified by the name of the segment on which the threshold is defined, and the current threshold free space setting in number of pages.

–freespace *num_pages* – specifies the new threshold limit in number of pages. When the space available on the segment drops below this limit, the threshold is triggered.

–sproc *proc_name* – specifies the name of the stored procedure to execute when the threshold is reached.

–database *db_path* – specifies the database that contains the segment on which the threshold exists. The default is the current database collection (see “Command Context” on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
ssetthresh –name "logseg 256" –sproc log_thresh
```

Modifies the threshold on the log segment that has a limit of 256 pages to execute the procedure **log_thresh** when the threshold is reached.

Comments

- Before using **ssetthresh**, you need the following information:
  - The database containing the segment on which the threshold exists.
  - The name of the segment and the page limit at which the threshold is defined. These two pieces of information uniquely identify a threshold.
- The procedure, or other specified event manager, does not have to exist at the time the threshold is modified.

Permissions

To use **ssetthresh**, you must have the following roles:

TME	**ESSM**	**SQL Server**
any	**space**	**sa_role**

See Also

scrtthresh, scompseg, scopyseg, sdelthresh, sgetseg

ssetuser

Function

Modifies a database user’s attributes, including group membership and login aliases.

Syntax

```
ssetuser –name user_name [–group group_name]
   [–addaliases login_names]
   [–dropaliases login_names] [–database db_path]
   [–version] [–help]
```

Parameters

–name *user_name* – specifies the user to modify.

–group *group_name* – specifies the group that the user is to belong to. To remove a user from a group, specify “public” as the group name. A user can be a member of only one group in addition to the group, “public.”

–addaliases *login_names* – specifies the logins to add as aliases for the specified user in the database.

–dropaliases *login_names* – specifies the aliases to remove for the specified user.

–database *db_path* – specifies the name of the user’s database. The default is the current database collection (see “Command Context” on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1.
   ```
   ssetuser –n webster –group jazz –database
   KOKO/pubs2
   ```

   Changes group of user *webster* to *jazz*.

2.
   ```
   ssetuser –n guest –addalias barneyb,bwebster
   –dropalias probe
   ```

   Maps logins *barneyb* and *bwebster* as aliases for user *guest*, and removes the alias login *probe* from user *guest*.

Comments

- Before using **ssetuser**, you need the following information:
  - The database to access
  - The user name
- Groups or aliases must already exist before you can assign them to the user.

Permissions

To use **ssetuser**, you must have the following roles:

TME	ESSM	SQL Server
any	**security**	**sa_role** or Database Owner

See Also

scrtuser, scrtgroup, scrtlogin, scompuser, scopyuser, sdeluser, ssetgroup

sstartserver

Function

Starts up a managed SQL Server.

Syntax

```
sstartserver [–name server_name]
   [–file runserver_file] [–single] [-backupserver]
   [–version] [–help]
```

Parameters

–name *server_name* – identifies the managed SQL Server to start. The default is the current managed SQL Server collection (see "Command Context" on page 1-15).

–file *runserver_file* – identifies the full path name of a runserver file.

–single – starts SQL Server in single user mode, allowing only one System Administrator to log in. This mode is used for restoring the *master* database. The default is multi–user mode.

–backupserver – specifies that this server is being started as a backup server.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
sstartserver –name COPY –f
/usr/local/sybase/install/RUN_TEST
```

Starts the Backup Server COPY using the specified runserver file *RUN_TEST*.

Comments

- Before using **sstartserver**, you need the following information:
  - The name of the managed SQL Server you want to start
  - The full path name of the runserver file to use

- This command creates or appends to the SQL Server *errorlog* file. You can look at this file to view the start–up messages for SQL Server.
- This command does not wait for SQL Server to fully start.

Permissions

To use **sstartserver**, you must have the following roles:

TME	ESSM	SQL Server
any	**server**	**sa_role**

See Also

smanageserver, sunmanageserver, scompserver, scopyserver, sgetserver, sgetprocess, skillprocess, sstopserver

sstopserver

Function

Shuts down a managed SQL Server.

Syntax

```
sstopserver [–name server_name]
   [–restart | –immediate] [–version] [–help]
```

Parameters

–name *server_name* – specifies the managed SQL Server to stop. The default is the current managed SQL Server collection (see "Command Context" on page 1-15).

–restart – restarts SQL Server after shutdown.

–immediate – stops SQL Server immediately. If you do not specify this option, the command waits for existing processes to exit before stopping SQL Server.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Examples

1. **`sstopserver –name BOSTON`**

   Shuts down the SQL Server BOSTON, waiting for existing processes to exit. Does not restart SQL Server.

2. **`sstopserver –name COPY –immed`**

   Shuts down the Backup Server COPY immediately without waiting for processes to exit. Does not restart SQL Server.

Comments

Before using **sstopserver**, you need to know the following:

- The name of the managed SQL Server you want to stop.
- Whether there are processes running that you would like to finish before stopping SQL Server.

Permissions

To use **sstopserver**, you must have the following roles:

TME	ESSM	SQL Server
any	**server**	**sa_role**

See Also

smanageserver, sunmanageserver, scompserver, scopyserver, sgetserver, sgetprocess, skillprocess, sstartserver

ssyncprf

Function

Synchronizes the contents of a profile with its associated elements in the associated SQL Server or database. Use this command to track changes made to a SQL Server or database through tools other than Enterprise SQL Server Manager. Running this command prior to distributing a profile can insure that all changes made to profiled elements, via **isql** or some other tool, will be distributed to the subscribers.

Syntax

```
ssyncprf -name profile_name -type type [-version] [-
   help]
```

Parameters

–name *profile_name* – specifies the name of the profile to synchronize.

–type *type* – specifies the type of the profile to synchronize.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – Prints the usage statement for this command.

Comments:

This command will "touch" each element of the profile that still exists in the associated SQL Server or Database. This causes all elements to be distributed at the next distribution. Elements that are in the profile but have been deleted via **isql** are marked as deleted in the profile so that their deletion also gets distributed to subscribers.

Example:

```
ssyncprf -name login_profile -type SQLLoginProfile
```

This example synchronizes the profile named *login_profile* that has the type *SQLLoginProfile*.

Permissions

To use **ssyncprf**, you must have the following roles:

TME	ESSM	SQL Server
senior or **super**	none	**sso_role**

See Also

`scrtprfmgr, sgetprf, spopulateprf`

ssyncserver

Function

Updates the list of managed databases within the managed SQL Server.

Syntax

```
ssyncserver [-name server_name] [–version] [–help]
```

Parameters

–name *server_name* – specifies the managed SQL Server to be updated. The default is the current managed SQL Server collection (see "Command Context" on page 1-15).

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
ssyncserver -name CONCORD
```

Updates the list of managed databases in SQL Server CONCORD.

Comments

- Before using **ssyncserver**, you must know the name of the managed SQL Server.
- If a database was created outside of Enterprise SQL Server Manager (for example, using **isql**) after SQL Server was managed by Enterprise SQL Server Manager, the database will not automatically appear in the Enterprise SQL Server Manager SQL Server collection. The **ssyncserver** command adds such databases to the list of managed databases and includes them in the SQL Server collection.
- If a database was deleted outside of Enterprise SQL Server Manager (for example, using **isql**) after SQL Server was managed by Enterprise SQL Server Manager, the database will not automatically disappear from the Enterprise SQL Server Manager SQL Server collection. The **ssyncserver** command removes such databases from the list of managed databases and from the SQL Server collection.

Permissions

To use **ssyncserver**, you must have the following roles:

TME	ESSM	SQL Server
any	none	none

See Also

smanageserver, ssetserver, sgetserver

sunmanageserver

Function

Removes a SQL Server's registration as a managed server. Once removed, the server can no longer be administered using Enterprise SQL Server Manager.

Syntax

```
sunmanageserver –name server_name [–version] [–help]
```

Parameters

–name *server_name* – specifies the managed SQL Server to stop managing.

–version – prints the release number and copyright date of the Enterprise SQL Server Manager software that you are using.

–help – prints the usage statement for this command.

Example

```
    sunmanageserver –name KOKO
```

Removes the registration of the SQL Server KOKO as a managed Server.

Comments

- Before using **sunmanageserver**, you must know the name of the managed SQL Server.
- It is recommended that the host that has SQL Server on it be the management host.

Permissions

To use **sunmanageserver**, you must have the following roles:

TME	ESSM	SQL Server
server	none	none

See Also

smanageserver, ssetserver, sgetserver, sstartserver, sstopserver

A Database Options

The Database Options

These are the database options that you can set:

abort tran on log full
allow nulls by default
auto identity
dbo use only
ddl in tran
no chkpt on recovery
no free space acctg
read only
select into/bulkcopy
single user
trunc log on chkpt

The following sections describe the database options.

abort tran on log full

abort tran on log full determines the fate of a transaction that is running when the last-chance threshold is crossed. The default value is **false**, meaning that the transaction is suspended and is awakened only when space is freed. If you change the setting to **true**, all user queries that need to write to the transaction log are killed until space in the log is freed.

allow nulls by default

Setting **allow nulls by default** to **true** changes the default null type of a column from **not null** to **null**, in compliance with the ANSI standard. The Transact-SQL default value for a column is **not null**, meaning that null values are not allowed in a column unless **null** is specified in the column definition.

auto identity

When the **auto identity** option is set to **true**, a 10-digit IDENTITY column is automatically defined in each new table in which neither a **primary** key, **unique** constraint, nor IDENTITY column is specified. The column is not visible when you select all columns with the **select *** statement. To retrieve it, you must explicitly mention the column name, SYB_IDENTITY_COL, in the select list.

dbo use only

While the **dbo use only** option is set to **on** (or **true**), only the Database Owner can use the database.

ddl in tran

Setting the **ddl in tran** option to **true** allows the following commands to be used inside a user-defined transaction:

Table A-1: DDL commands allowed in transactions

alter table	**create table**	**drop rule**
create default	**create trigger**	**drop table**
create index	**create view**	**drop trigger**
create procedure	**drop default**	**drop view**
create rule	**drop index**	**grant**
create schema	**drop procedure**	**revoke**

Data definition statements lock system tables for the duration of a transaction which can result in performance problems.

The following commands cannot be used inside a user-defined transaction under any circumstances:

Table A-2: DDL commands not allowed in transactions

alter database	**load database**	**truncate table**
create database	**load transaction**	**update statistics**
disk init	**reconfigure**	
drop database	**select into**	

no free space acctg

no free space acctg suppresses free space accounting and the execution of threshold actions for the non-log segments. This speeds recovery time because the free-space counts are not recomputed for those segments. It also disables updating the rows-per-page value stored for each table with the result that system procedures that estimate space usage may report inaccurate values.

no chkpt on recovery

The **no chkpt on recovery** option is set to **on** (**true**) when an up-to-date copy of a database is kept. In these situations, there is a primary and a secondary database. Initially, the primary database is dumped and loaded into the secondary database. At intervals, the transaction log of the primary database is automatically dumped and loaded into the secondary database.

If this option is set to **off** (**false**)—the default condition—a checkpoint record is added to the database after it is recovered due to restarting the server. This checkpoint ensures that the recovery mechanism will not be re-run unnecessarily and changes the sequence number on the database. If the sequence number on the secondary database changes, a subsequent dump of the transaction log from the primary database cannot be loaded into it.

Turning on this option for the secondary database prevents it from getting a checkpoint from the recovery process so that subsequent transaction log dumps from the primary database can be loaded into it.

read only

The **read only** option means that users can retrieve data from the database but cannot modify anything.

select into/bulkcopy

The **select into/bulkcopy** option must be on in order to perform operations that do not keep a complete record of the transaction in the log:

- Use the **writetext** utility
- **select into** a permanent table
- Do a “fast” **bulk copy** with **bcp**. “Fast” **bcp** is used by default on tables that do not have indexes.

SQL Server performs minimal logging for these commands, recording page allocations and de-allocations but not the actual changes that are made on the data pages.

You do not have to turn on the **select into/bulkcopy** option in order to **select into** a temporary table, since the temporary database is never recovered. Likewise, the option does not need to be on in order to run **bcp** on a table that has indexes, because inserts are logged.

With select **into/bulk copy** turned on, you cannot dump the transaction log using the **dump transaction** statement. You can still use **dump transaction...with no_log** and **dump transaction...with truncate_only.**

By default, the **select into/bulkcopy** option is off in newly created databases. To change the default situation, turn on this option in the *model* database.

single user

When **single user** is set to **true**, only one user at a time can access the database.

trunc log on chkpt

The **trunc log on chkpt** option means that the transaction log is truncated (committed transactions are removed) every time the **checkpoint** checking process occurs (usually more than once per minute). When the Database Owner runs **checkpoint** manually, however, the log is **not** truncated.

It may be useful to turn on this option while doing development work during which backups of the transaction log are not needed. If this option is off (the original default condition) and the transaction log is never dumped, the transaction log continues to grow and you may run out of space in your database.

When the **trunc log on chkpt** option is on, you cannot dump the transaction log because changes to your data are not recoverable from transaction log dumps. In this situation, issuing the **dump transaction** command produces an error message instructing you to use **dump database** instead.

By default, the **trunc log on chkpt** option is off in newly created databases. To change the default situation, turn on this option in the *model* database.

◆ ***WARNING!***

If you turn on the trunc log on chkpt option in the *model* database and need to load a set of database and transaction logs into a newly created database, be sure to turn off the option in the new database to avoid truncating the transaction log and to enable dumping the transaction log.

For a report on which database options are set in a particular database, execute the system procedure **sp_helpdb** in that database. See *Enterprise SQL Server Manager User's Guide* for additional information on using database options.

Glossary

alias

Login name mapped to one or more users in addition to their personal login. When a user logs in using an alias, the user gains the privileges assigned to that login.

auditing

ESSM provides the ability to record and produce reports on security-related server and database activity. The information is recorded in a traceable audit trail.

audit trail (system table *sysaudits*)

The *sysaudits* table contains one row for each audit record in the table.

authentication

ESSM includes commands and a graphical user interface for creating and verifying the identity of logins, database users, and remote SQL Servers.

authorization

ESSM provides lets you assign roles to specific users that bestow automatic privileges. ESSM also lets you grant and revoke permission to users, groups, and roles to control use of specific commands and objects in a database.

backup

A copy of a database or transaction log, used to recover from a media failure.

batch

One or more Transact-SQL statements submitted as a group to SQL Server for processing and terminated by an end-of-batch signal.

bulk copy

The utility for copying data in and out of databases, called **bcp**.

cache

A storage buffer used to reduce access time for frequently accessed data or procedures.

character set

A set of specific characters with an encoding scheme that uniquely defines each character. ASCII is a common character set.

checkpoint

The point at which all changed data pages are guaranteed to be written to the database device.

CLI (command line interface)

ESSM offers a complete set of commands for managing SQL Servers and databases in the enterprise. These commands are compatible with the Tivoli commands and can be entered from the UNIX command line or from scripts. Scripts can be defined as tasks in the Tivoli task library and scheduled as jobs in the Tivoli scheduler.

collection

A customized grouping of SQL Server and database objects. You can organize objects into groups that are most useful to you, such as a collection of database users for each department. You can put collections within collections.

command

A statement that instructs the computer to perform an operation. The command begins with a word, often a verb, that names the operation. In addition, the command can include one or more keywords with or without variable values that tailor the command.

command permissions

Permission to use specific commands. Some command permissions are implicitly granted by a database user’s role, other command permissions are explicitly granted with the **grant** command. See also **object permissions**.

database

A set of related data tables with a given structure for accepting, storing, and providing data for one or more users. Objects relative to using the database, such as users, indexes, rules, and so on are also considered part of the database.

database device

A device dedicated to the storage of the objects that make up databases. A device can be all or part of a disk or file used to store databases and database objects.

database object

One of the components of a database: table, view, index, procedure, trigger, column, default, or rule.

database owner

The system administrator who creates a database is automatically the Database Owner. The system administrator can assign ownership of a database to another user. The owner controls all the objects in the database. In a user's own database, SQL Server recognizes the user as *dbo*.

default

An attribute, condition, value, or option that is assumed when none is explicitly specified.

default database

The database that a user gets by default when he or she logs in.

default language

The language used to display prompts and messages within SQL Server.

disk mirror

A duplicate SQL Server database device. All writes to the primary device are copied (mirrored) to a separate secondary device. Writes can be either serial (consecutive) or parallel (simultaneous). If one device fails, the other contains an up-to-date copy of all transactions.

distributed computing environment

An environment comprising a variety of platforms and applications connected by one or more networks.

distribution

The process of copying SQL Server or database information to other managed SQL Servers or profile managers in the distributed computing environment.

dump volume

A single tape, partition, or file used for a database or transaction dump. A dump can span many volumes, or many dumps can be made to a single volume.

enterprise

See **distributed computing environment**.

error message

A message that SQL Server issues, usually to the user's terminal, when it detects an error condition.

ESSM (Enterprise SQL Server Manager)

An application that provides centralized control of SQL Server installations in a distributed computing environment. ESSM runs within the Tivoli Management Environment (TME).

ESSM administrator

A system administrator managing SQL Server installations within the TME enterprise. The administrator must have appropriate SQL Server roles, ESSM roles, and TMR roles. See **roles**.

events

See **Event Monitoring Services**.

Event Monitoring (EMON) Services

The user interface that lets you monitor generic events, process events, and error events. You can specify levels at which you are notified of an event and a variety of actions that are triggered by an event.

free-space threshold

A user-specified threshold that specifies the amount of space on a segment, and the action to be taken when the amount of space available on that segment is less than the specified space.

GUI (graphical user interface)

ESSM provides a windows environment with menus, icons, and dialog boxes for managing SQL Server installations and databases in a distributed computing environment. These graphic elements are compatible with and exist within the Tivoli Desktop.

high-level event service

See **Event Monitoring Services**.

locking

The process of restricting access to resources, such as logins or system tables, in a multi-user environment to maintain security and prevent concurrent access problems.

login

A login consists of a login name and a password that gives a user access to a specific system, SQL Server, database, and so forth. In addition, each ESSM administrator has a SQL Server login account.

managed server

A SQL Server that is in a policy region's managed resource list.

managed resource

A specific resource that has a default policy defined in a policy region. For example, a SQL Server can be defined as a managed resource in a TME policy region. A specific managed resource can belong to only one policy region at a time. Each resource is one of several types. The policy region must contain the resource type in its type list before a specific resource can be added to the policy region.

***master* database**

Controls the user databases, system tables, and the operation of SQL Server as a whole. This database keeps track of such things as user accounts, ongoing processes, and system error messages.

mirror

See **disk mirror**.

***model* database**

The default template for a new SQL Server database. Each time a database is created, SQL Server makes a copy of *model* and extends it to the size requested.

notices

A message concerning some operation or change in the distributed system. Messages can be notices on the TME Bulletin Board, e-mail messages, pop-up dialog boxes, and so on.

null

Having no explicitly assigned value. Null is not equivalent to zero nor to blank. A value of null is not greater than, less than, or equivalent to any other value, including another value of null.

object permissions

Permission to use and modify specific tables, views, or procedures (such as permission to update a table). Some object permissions are implicitly granted by a database user's role, other object permissions are explicitly granted with the **grant** command. See also **command permissions**.

oper

The SQL Server operator role, **oper_role**. See **roles**.

parameter

A variable value used in conjunction with a command or a stored procedure. Also a keyword and value that defines a SQL statement.

permission

See **command permissions**, **object permissions**, and **roles**.

policy region

A collection of TME resources that are governed by a common set of policies. ESSM administrators are given the authority to manage resources in one or more policy regions. A policy region contains a list of resource types that are valid for that policy region. You can add or remove resource types from the list so that you can control the kinds of resources the policy region will govern.

policy

Rule that governs the management of resources, such as requiring login accounts to have passwords. ESSM policy methods take the form of shell scripts. Default policy methods govern default characteristics of resources. Validation policy methods protect the integrity of resources.

profile

A collection of object-specific information. Each profile is one of several types. There are nine ESSM profile types. For example, a user profile might contain information on the name, login, group, permissions, and ownership for each user defined in the profile. Profile information can be distributed across an environment that contains multiple platforms.

profile endpoint

A SQL Server or profile manager that subscribes to a profile; the endpoint is the final destination of data distributed from a profile manager.

profile manager

A window used to manage a group of profiles that are subscribed to as one unit by individual profile endpoints. An ESSM profile manager is associated with a specific SQL Server or a specific database. All profiles in the profile manager relate to the associated SQL Server or database.

recovery

The process of rebuilding one or more databases from database dumps and transaction log dumps.

remote procedure call (RPC)

A stored procedure executed in a different SQL Server from the server the user is logged into.

resource

A system, device, service, or facility in a distributed system. For example, file systems, workstations, administrators, and SQL Server installations can be resources in the Tivoli Management Environment. A directly managed resource is a resource that can be created and managed within a policy region. A validation resource has policy validation applied before each operation performed on the resource. You can modify a policy region's list of managed resource types.

roles

Roles assigned to ESSM administrators enable administrators to manage SQL Servers on the enterprise. There are three categories of roles to consider when using ESSM.

TMR roles	ESSM roles	SQL Server roles
user	space	sa_role
admin	server	sso_role
senior	load	oper_role
super	security	
restore	dump	
install_product	schema	
install_client		
backup		

An ESSM administrator must have the appropriate Tivoli, ESSM, and SQL Server roles to carry out a desired task. For example, an ESSM administrator must have both the ESSM **security** role and the SQL Server **sso_role** in order to create logins. All administrators must have the Tivoli **user** role to access their desktop.

sa

The login name for the SQL Server system administrator. Also the SQL Server system administrator role, **sa_role**. See **System Administrator** and see **roles**.

scheduler

The TME service that enables ESSM administrators with the TME **admin** role to schedule and run tasks and jobs. The scheduler is represented by a clock icon on the TME Desktop.

schema

As used in this manual, schema refers to database objects such as tables, procedures, views, and indexes.

segment

A named subset of database devices available to a particular database. A segment is a label that points to one or more database devices. Segments can be used to control the placement of tables and indexes on specific database devices.

source

The original SQL Server or database which is being copied or compared to another SQL Server or database. Also the SQL Server or database associated with a profile manager. Also the profile manager which is being distributed to another profile manager or SQL Server.

subscriber

A SQL Server or profile manager that receives profile information from a profile manager.

SQL Server

The server in the Sybase client/server architecture. SQL Server manages multiple databases and multiple users, keeps track of the actual location of data on disks, maintains mapping of logical data description to physical data storage, and maintains data and procedure caches in memory.

SQL Server login

See **login accounts**.

sso

The SQL Server system security officer role, **sso_role**. See **roles**.

statement

See **command**.

stored procedure

A collection of SQL statements and optional control-of-flow statements stored under a name. SQL Server-supplied stored procedures are called system procedures.

Sybase login account

See **login accounts**.

System Administrator

A user in charge of SQL Server system administration, including creating user accounts, assigning permissions, and creating new databases. See also **ESSM administrator**.

system databases

The four databases in a newly installed SQL Server: the master database (*master*), that controls user databases and the operation of the SQL Server; the temporary database (*tempdb*) used for temporary tables; the system procedures database (*sybsystemprocs*); and the model database (*model*) used as a template to create new user databases.

system table

The system tables keep track of information about SQL Server as a whole and about each user database. The *master* database contains some system tables, such as *sysalternates*, *sysaudits*, *sysconfigures*, and *syscurconfigs*, that are not in user databases.

target

The SQL Server or database to which the source server or database is being copied or compared. Also the profile manager or SQL Server that is receiving distributed information from a source profile manager.

task library

The Tivoli Management Environment lets you create a task library in which you can create and store tasks and jobs. These tasks and jobs can be run immediately or scheduled to run at a specific time. A task is a TME resource that encapsulates daily operations, such as clearing the printer queue. Jobs are created from tasks. A job lets you specify details of task execution, such as where to display output.

temporary database

The temporary database in SQL Server, *tempdb*, that stores temporary tables and other temporary data such as intermediate results of **group by** and **order by.**

threshold

A limit to monitor space usage on a database segment. When the available space is below the limit, SQL Server executes a user-specified stored procedure.

Tivoli Name Registry

Contains the name and object ID of all objects in the TME database, including the SQL Server login name (same as the UNIX login name) and encrypted password information for each ESSM administrator.

TME (Tivoli Management Environment)

A software environment providing centralized control of integrated software products for a distributed system.

TME Desktop

The window containing menu bars and icons that let you visualize and control the various elements of the distributed environment.

TMP (Tivoli Management Platform)

The foundation for managing resources in a distributed environment. The TMP is the runtime platform for TME applications and provides the administrator with a view into the network and a set of tools that are applicable across functions and applications in the TME.

TMR (Tivoli Management Region)

A TME server and the set of clients it serves. A single TMR can contain a maximum of 200 clients. If your network contains more than 200 clients, you can create several TMRs and connect them.

transaction

A mechanism for ensuring that a set of actions is treated as a single unit of work.

transaction log

A system table (*syslogs*) in which all changes to the database are recorded.

username

The name by which a login is known in a database.

view

An alternative way of looking at the data in one or more tables. Usually created as a subset of columns from one or more tables.

wildcard

A special character used to represent one or more characters in a pattern-matching string. Any character or set of characters can replace a wildcard character. Use the **–wildcard** option, in commands that support it, to specify wildcard names.

Index