

# 4. Database Administration

---

## 4.1 Overview of Database Administration

The Database Administrator or DBA, is the individual or group responsible for the installation, configuration, update, maintenance, and overall integrity, performance and reliability of the SQL Server database. In general, the DBA is concerned with the availability of the server, the definition and management of resources allocated to the server, the definition and management of databases and objects resident on the server, and the relationship between the server and the operating system.

Each section in this chapter provides necessary background information, followed by step-by-step instructions and actual scripts where applicable.

## 4.2 SQL Server Environment

### 4.2.1 Naming Conventions,

As one of the most important, yet least applied concepts, naming conventions are presented in this chapter by examples according to the following rules.

**Rule 1:** Regardless of the length of the name, it should indicate the function and/or content of the object

**Rule 2:** Only easily understandable abbreviations should be used

**Rule 3:** Parts of names are separated by underscores “\_”, only one optional suffix is permitted (appended to the name by a . “.”)

**Rule 4:** The full path of the object is considered to be part of the name

The names of the databases and tables themselves may or may not follow the above rules, these rules are specifically for the DBA to work with SQL Server objects, and files in the UNIX environment.

All **COTS** software is installed in the /usr/ecs/OPS/COTS directory.

All **SYBASE** software is located in the Sybase home directory (**\$SYBASE**).

All backups are located in **\$SYBASE/sybase\_dumps** directory, which may or may not be on a separate physical disk.

### Note

It is strongly recommended that backups be stored on a separate physical disk.

The database dumps are kept for a period of 2 days and also stored on a disk by Networker everyday. The database dumps are named as follows:

dbname.dat\_YYMMDDHHMM.Z

where MMDDHHMM is the “sortable” eight digit month, day, hour, and minute. For example, on the date this chapter was written, a backup directory called **backups\_for\_99021100024.Z**

All SQL script files have the extension **.sql** as a suffix. Their names reference the objects they create or functions they perform, and are all located in **\$SYBASE/scripts**.

SQL statement must follow precise syntactical and structural rules, and may include only SQL keywords, identifiers (names of databases, tables, or other database objects), operators, and constants. The characters that can be used for each part of a SQL statement vary from installation to installation and are determined in part by definitions in the default character set that version of the server uses.

For example, the characters allowed for the SQL language, such as SQL keywords, special characters, and Transact-SQL extensions, are more limited than the characters allowed for identifiers. The set of characters which may be used for data is much larger and includes all the characters that can be used for the SQL language or for identifiers.

The sections that follow describe the sets of characters that can be used for each part of a statement. The section on identifiers also describes naming conventions for database objects.

#### **4.2.1.1 SQL Data Characters**

The set of SQL data characters is the larger set from which both SQL language characters and identifier characters are taken. Any character in SQL Server's character set, including both single-byte and multibyte characters, may be used for data values.

#### **4.2.1.2 SQL Language Characters**

SQL keywords, Transact-SQL extensions, and special characters such as the comparison operators > and <, can be represented only by 7-bit ASCII values A- Z, a -z, 0-9, and the following ASCII characters:

#### **4.2.1.3 Identifiers**

Conventions for naming database objects apply throughout SQL Server software and documentation. Identifiers can be up to 30 bytes in length, whether or not multibyte characters are used. The first character of an identifier must be declared as an alphabetic character in the character set definition in use on Server.

The @ sign or \_ (underscore character) can also be used. The @ sign as the first character of an identifier indicates a local variable.

Temporary table names must either begin with # (the pound sign) if they are created outside tempdb or be preceded by "tempdb..".

Table names for temporary tables that exist outside tempdb should not exceed 13 bytes in length, including the number sign, since SQL Server gives them an internal numeric suffix.

After the first character, identifiers can include characters declared as alphabetic, numeric, or the character \$, #, @, \_, ¥ (yen), or £ (pound sterling). However, you cannot use two @@ symbols together at the beginning of a named object, as in "@@myobject." This naming convention is reserved for global variables, which are system-defined variables that SQL Server updates on an ongoing basis.

The case sensitivity of SQL Server is set when the server is installed and can be changed by a System Administrator. To see the setting for your server, execute this command: sp\_helpsort

#### **4.2.1.4 Delimited Identifiers**

Delimited identifiers are object names enclosed in double quotes. Using delimited identifiers allows you to avoid certain restrictions on object names. You can use double quotes to delimit table, view, and column names; you cannot use them for other database objects.

Delimited identifiers can be reserved words, can begin with non-alphabetic characters, and can include characters that would not otherwise be allowed. They cannot exceed 28 bytes.

Before creating or referencing a delimited identifier, you must execute:

```
set quoted_identifier on
```

The names of database objects need not be unique in a database.

However, column names and index names must be unique within a table, and other object names must be unique for each owner within a database. Database names must be unique on SQL Server.

If you try to create a column using a name that is not unique in the table or to create another database object such as a table, a view, or a stored procedure, with a name that you have already used in the same database, SQL Server responds with an error message.

You can uniquely identify a table or column by adding other names that qualify it, that is, the database name, the owner's name, and, for a column, the table name or view name. Each of these qualifiers is separated from the next by a period:

```
database.owner.table_name.column_name
```

```
database.owner.view_name.column_name
```

The same naming syntax applies to other database objects. You can refer to any object in a similar fashion:

If the quoted\_identifier option of the set command is on, you can use double quotes around individual parts of a qualified object name.

Use a separate pair of quotes for each qualifier that requires quotes.

For example, use:

database.owner."table\_name"."column\_name"

rather than:

database.owner."table\_name.column\_name"

The full naming syntax is not always allowed in create statements because you cannot create a view, procedure, rule, default, or trigger in a database other than the one you are currently in. The naming conventions are indicated in the syntax as:

[[database.]owner.]object\_name or: [owner.]object\_name

The default value for owner is the current user, and the default value for database is the current database. When you reference an object in SQL statements, other than create statements, without qualifying it with the database name and owner name, SQL Server first looks at all the objects you own, and then at the objects owned by the Database Owner, whose name in the database is "dbo." As long as SQL Server is given enough information to identify an object, you need not type every element of its name. Intermediate elements can be omitted and their positions indicated by periods:

database..table\_name

You must include the starting element, in this case, database, particularly if you are using this syntax when creating tables. If you omit the starting element, you could, for example, create a table named ..mytable. This naming convention prevents you from performing certain actions on such a table, such as cursor updates.

When qualifying a column name and a table name in the same statement, be sure to use the same naming abbreviations for each; they are evaluated as strings and must match or an error is returned.

#### **4.2.1.5 Identifying Remote Servers**

You can execute stored procedures on a remote SQL Server, with the results from the stored procedure printed on the terminal that called the procedure. The syntax for identifying a remote server and the stored procedure is:

[execute] server.[database].[owner].procedure\_name

You can omit the execute keyword when the remote procedure call is the first statement in a batch. If other SQL statements precede the remote procedure call, you must use execute or exec. You must give the server name and the stored procedure name. If you omit the database name, SQL Server looks for procedure\_name in your default database. If you give the database name, you must also give the procedure owner's name, unless you own the procedure or the procedure is owned by the Database Owner.

If the server name in interfaces is in uppercase letters, you must use it in uppercase letters in the remote procedure call.

In all cases throughout this chapter, when actual examples are provided, those which reference UNIX commands will be preceded by a “%”, and those that reference SQL statements will be preceded by a number and a “>” (e.g. 1>sp\_help tablename).

The terms described in the following table will be used throughout this chapter.

**Table 4.2-1. SQL Server General Definitions**

Term	Definition
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.
Client	SYBASE Open Client software located in the /tools/sybOCv(TBD) directory for SUN and HP platforms SYBASE Open Client software located in the /tools/sybOCv(TBD) directory for SGI platform
Backup Server	Similar to the dataserver, it uses a separate UNIX process to off load the cycles associated with DUMP and LOAD commands
backups	The set of UNIX files containing full database dumps, transaction log dumps, and dbcc output
dbcc	Database Consistency Checker - a utility program designed to check the logical and physical consistency of a database
sybase root directory	/usr/ecs/OPS/COTS/sybase, this is the home directory for all SYBASE software and related products and is referenced both in UNIX and in the rest of this document as <b>\$SYBASE</b>
interfaces file	Lists the names and access paths for all servers and backup servers. This file is located in the <b>\$SYBASE</b>
sa	System Administrator login, this is the superuser of the SQL Server
scripts	UNIX script programs located in <b>\$SYBASE/scripts</b> and related subdirectories {\${Secs_Home}},{mode}/custom/dbms/{subsystem}
showserver	A utility invoked at the UNIX command prompt to display active servers, located in <b>\$SYBASE/install</b> .
SQL scripts	SQL and command statements located in <b>\$SYBASE/scripts</b> and related subdirectories and /{\${Secs_Home}},{mode}/custom/dbms/{subsystem}
Server Name	The name of the database server for a specific application in different modes <b>EX.</b> - PDPS application database server in OPS mode EX.- Pdps_TS1 in TS1 mode EX. -pdps_TS2 in TS2 mode
Port Numbers	The port number to be utilized by the above listed servers.
Release Directory	<b>\$SYBASE</b>
SQL	Structured Query Language

#### 4.2.2 SQL Server Directory Structure

The **sybase** directory structure is described in the following table. Subdirectories under the **scripts** can contain template files with easy to modify examples of SQL and SQL command syntax.

**Table 4.2.2-1. SYBASE Directory Structure**

Directory	Contains
<b>\$SYBASE/bin</b>	Utilities necessary to load, run, and access the server
<b>\$SYBASE /install</b>	Files used to start and initialize dataservers, backupserver and to record server messages (errorlogs)
<b>\$SYBASE /lib</b>	db-lib, ct-lib, and xa-lib client library files used by applications to gain access to the server (local to server) <b>*Applications use automounted libraries.</b>
<b>\$SYBASE /scripts</b>	Root directory for all script files executed on the server
<b>\$SYBASE /sybase_dumps</b>	Root directory that contains all backup subdirectories, it is recommended, but not required, that this directory be on a separate physical disk. Dumps both database and transaction logs. <b>**Backups are stored on disk in the backup subdirectories.</b>
backup subdirectories \$SYBASE /sybase_dumps/dumps \$SYBASE /sybase_dumps/trans \$SYBASE /sybase_dumps/dumps/logs \$SYBASE /sybase_dumps/trans/logs \$SYBASE /sybase_dumps/Week1 \$SYBASE /sybase_dumps/Week2 \$SYBASE /sybase_dumps/Week1/logs \$SYBASE /sybase_dumps/Week2/logs	A cron job is run at night to move data from the current (week1) directory to the previous (week2) directory. Then, a dump of the databases and transaction logs is executed and is stored in the current directory. All logs are written to the log directory. Files are saved using the following naming convention:: dbname.dat.YYMMDDHHMM.Z - full database dumps dbname.tran.YYMMDDHHMM.Z - full transaction log dumpsdbname_backup.log. dbname_ERR.log.MMDDHHMM - Error log filesdbname_dbcc.log. MMDDHHMM
** <u>xx</u> dmh02 serves as a remote Backup Server	**xx are the 2 letter codes to identify a DAAC site (i.e., g0 = Goddard)

### 4.2.3 SQL Server Installation

SYBASE SQL Server Version (TDB) has been installed and configured by the ECS Installation Staff. Shared memory and disk resources have been allocated and configured by the System Administrator, and both the client and server portions have been set up by the DBA prior to shipment. The following table describes parameters and options used during installation.

**Table 4.2.3-1. SQL Server Parameters and Options**

Parameters Name	Brief Explanation/Settings
Retry Count	5 seconds
Retry Delay	5 seconds
Master device	28 Mb raw partition
Master Device Location	
Backup Server Name	SYB_BACKUP
sybssystemprocs	\$SYBASE/devices/(MachineName)_sybprocs.dat, 19 Mb and on it's own device
Errorlog	\$SYBASE/install/mode.errorlog ( <b>mode</b> indicates the application)
Current default language	us_english

Current default character set	iso_8859-1 (Latin-1)
Current sort order	Binary ordering, for the ISO 8859/1 or Latin-1 character set (iso_1).
Internal auditing	On
sybsecurity database size	175 Mb - Varies – depends on disk allocations
sybsecurity device	sybsecurity, positioned on a 175 Mb raw partition

The installation script files are located in the **\$SYBASE/install** directory. SQL Server installation is performed by an authorized user with the **sybinit** utility also located in the **\$SYBASE/install** directory. See your UNIX System Administrator and the SYBASE SQL Server Installation Guide.

### 4.3 Database Administrator Responsibilities

The following subsections detail the most common functions that a DBA will perform.

#### 4.3.1 Startup of SQL Server

Use **startserver** to start an SQL Server and/or a Backup Server. This command can only be issued by the **Sybase** user.

Syntax:           % **startserver** [-f runserverfile]

The “runserverfile” is contained in the **\$SYBASE/install** directory.

#### Note

SQS server should be started after the SQL Server

#### 4.3.2 Shutdown of SQL Server

Use **shutdown** to bring the server to a halt. This command can only be issued by the Sybase System Administrator (sa).

Syntax: 1> **shutdown** [**backup\_server\_name**] [with] [wait] [with nowait]  
          2> **go**

The "with wait" is the default option. This option brings SQL Server down gracefully.

The “with nowait” option shuts down the SQL Server immediately without waiting for currently executing statements to finish.

If you do not give a server name, shutdown shuts down the SQL Server you are using.

When you issue a shutdown command, SQL Server:

1. Disables logins, except for System Administrators
2. Performs a checkpoint in each database, flushing pages that have changed from memory to disk
3. Waits for currently executing SQL statements or procedures to finish

In this way shutdown minimizes the amount of work that automatic recovery must do when you restart SQL Server.

To see the names of the Backup Servers that are accessible from your SQL Server, execute **sp\_helpserver**. Use the value in the name column in the shutdown command. You can only shut down a Backup Server that is:

- Listed in sys.servers on your SQL Server, and
- Listed in your local interfaces file.

#### Note 1

It is recommended that "with wait" option be used. This allows executing statements to finish. Also it is recommended that you perform a checkpoint of all database prior to shutdown.

#### Note 2

SQL server should be started after the SQL Server

### 4.3.3 Showing SQL Server(s)

Use **showserver** to determine whether the SQL Server(s) and/or Backup Server(s) are running.

Syntax:           % **showserver**

The "**showserver**" is contained in the \$SYBASE/install directory

**Example:** UNIX processes running the various servers:

UID	PID	PPID	C	STIME	TTY	TIME	COMD
sybase	671	669	80	Apr 17	?	80:05	/usr/ecs/OPS/COTS/Sybase/bin/dataserver -
							d /dev/rdisk/c1t0d0s1 -g0sps06_srvr
sybase	665	663	80	Apr 17	?	50:02	/usr/ecs/OPS/COTS/sybase/bin/backupserver
							-g0sps06_backup -e/usr/ecs/OPS

## 4.4 Allocation of Resources

SQL Server can make reasonable default decisions about many aspects of storage management, such as where databases, tables, and indexes are placed and how much space is allocated for each one. However, the System Administrator has ultimate control over the allocation of disk resources to SQL Server and the physical placement of databases, tables, and indexes on those resources.

### 4.4.1 Creating Logical Devices

A logical device is created when the UNIX System Administrator determines that new disk space is available for use by SYBASE software, databases, transaction logs, and/or backups. Either raw disk partitions or UNIX filesystem partitions can be used to create a logical device. The creation of a logical device is a mapping of physical space to a logical name and virtual device number (**vdevno**) contained in the SQL Server **master** database. The **disk init** command is used to initialize this space. After the disk initialization is complete, the space described by the physical address is available to SQL Server for storage, and a row is added to the **sysdevices** table in the **master** database.

A System Administrator initializes new database devices with the disk init command.

Disk Init does the following: Maps the specified physical disk device or operating system file to a database device name

Lists the new device in master..sysdevices

Prepares the device for database storage

#### Note

Before you run disk init, see the SQL Server installation and configuration guide for your platform for information about choosing a database device and preparing it for use with SQL Server. You may want to repartition the disks on your computer to provide maximum performance for your Sybase databases.

Disk init divides the database devices into allocation units of 256 2K pages, a total of 1/2MB. In each 256-page allocation unit, the disk init command initializes the first page as the allocation page, which will contain information about the database (if any) that resides on the allocation unit.

#### Note

After you run the disk init command, be sure to use dump database to dump the master database. This makes recovery easier and safer in case master is damaged. If you add a device and fail to back up master, you may be able to recover the changes with disk reinit.

Syntax: **disk init**

name = "device\_name" ,

```
physname = "physicalname" ,  
vdevno = virtual_device_number ,  
size = number_of_blocks  
[, vstart = virtual_address ,  
  cntrltype = controller_number]
```

#### 4.4.1.1 Example of Creating a Logical Device

A raw partition on a RAID device has been made available to SQL Server by the UNIX System Administrator. Essentially, the actual name of the raw device **c2t0d1s3** has had its ownership changed to **sybase** and its group changed to **user**.

1. In **\$\$SYBASE/scripts/create.devices**, DBA makes a script file from the template.

```
Syntax: % cd /usr/ecs/OPS/COTS/sybase/scripts/create.devices
```

```
        % cp template.sql data_dev1.sql
```

2. Appropriate items are modified so that the script file resembles the following:

```
1> disk init
```

```
2> name = "data_dev1",
```

```
3> physname = "/dev/rdisk/c2t0d1s3",
```

```
4> vdevno = 3,
```

```
5> size = 128000
```

```
6> go
```

```
7> sp_helpdevice data_dev1
```

```
8> go
```

3. DBA runs the script from the UNIX command prompt:

```
Syntax:      % isql -Usa -Sservername -idata_dev1.sql -odata_dev1.out
```

4. DBA checks the **data\_dev1.out** file for success

#### 4.4.2 Creating and Altering Databases

A user database is created by the DBA with a script containing the **create database** command. A database is created on one or more physical devices. Specifying the device is optional - but highly recommended. When indicating the device, you use the logical name you specified as part of a

**disk init** (described above). Unlike the **disk init** command, the size of the database data and log components is specified in MB instead of 2K pages.

#### 4.4.2.1 Example of Creating a Database

The logical device **data\_dev1** has been created (as above) along with another device called **tx\_log1** (for transaction logging).

1. In **\$\$SYBASE/scripts/create.databases** directory, DBA makes a script file from the template.

```
Syntax: % cd /usr/ecs/OPS/COTS/sybase/scripts/create.databases
        % cp template.sql userdb.sql
```

2. Appropriate items are modified so that the script file resembles the following:

```
1> create database UserDB on data_dev1 = 100 log on tx_log1 = 50 [with override]
2> go
3> sp_helpdb UserDB
4> go
```

3. DBA runs the script from the UNIX command prompt:

```
Syntax:      % isql -Usa -Sservername -iuserdb.sql -ouserdb.out
```

4. DBA checks the userdb.out file for success

#### 4.4.2.2 Example of Altering a Database

The user database **UserDB** has run out of space and it has been determined that it should be increased by 50MB.

1. In **\$\$SYBASE/scripts/create.databases**, DBA creates a script file containing the ALTER DATABASE command (named alter\_userdb.sql)

```
Syntax: Alter database UserDB on data_dev3 = 50
```

2. DBA runs the script from the UNIX command prompt:

```
Syntax:      % isql -Usa -Sservername -ialter_userdb.sql -oalter_userdb.out
```

3. DBA checks the alter\_userdb.out file for success

#### 4.4.2.3 Data Placement - Segmentation

Segments are named subsets of the database devices available to a particular SQL Server database. Segment names are used in **create table** and **create index** commands to place tables or

indexes on specific database devices. Using segments allows the DBA to better control the size of database objects and may improve performance by spreading i/o more evenly across devices.

Once the database device exists and is available, the segment can be defined with the system stored procedure **sp\_addsegment**.

Syntax: `sp_addsegment segname, dbname, devname`

After the segment has been defined in the current database, the **create table** or **create index** commands use the optional clause “on segment\_name” to place the object on a particular segment.

Syntax: `create table table_name (column_name datatype ...) [on segment_name]  
create [clustered | nonclustered] index index_name on table_name (columns)`

Use **sp\_helpdb** database\_name to display the segments defined for that database.

Use **sp\_helpsegment** segment\_name to list the objects on the segment and show the mapped devices.

#### 4.4.2.3.1 Example of Creating a Segment

The DBA receives a request to create a segment for the storage of the DATA\_INFO table indexes in the pdps\_db\_ops database, on a separate physical disk. Two devices **data\_dev1** and **data\_dev2** have already been created and are located on different physical disks.

1. In `$$SYBASE/scripts/create.segments` directory, DBA makes a script file from the template.

Syntax: `% cd /usr/ecs/OPS/COTS/sybase/scripts/create.segments`

`% cp template.sql segments_dev1.sql`

2. The script file is modified so that it resembles the following:

```
1> sp_addsegment seg1_dev1, pdps, data_dev1
```

```
2> sp_addsegment seg1_dev2, pdps, data_dev2
```

```
3> go
```

3. DBA runs the script from the UNIX command prompt:

Syntax: `% isql -Usa -Sservername -ipdps_db_ops_segments.sql \  
-opdps_db_ops_segments.out`

4. DBA checks the `pdps_segments.out` file for success

5. When the table and indexes are created according to the instructions in section 4.4.6, the “on seg1\_dev1” must be appended to the DATA\_INFO.sql **create table** statement,

and the “on seg1\_dev2” must be appended to the DATA\_INFO\_indexes.sql CREATE INDEX statement.

Syntax: **create index** DATA\_INFO\_IDX on DATA\_INFO (DI\_ID) on SEG1\_DEV2

#### 4.5 Loading a database you have created into a different database:

Occasionally, you may want to create an exact copy of a database of your system. First, dump the existing database. Then create a database to load with this dump. The database does not have to be the same size as the original. The only requirement is that the destination database must be at least as large as the dumped database and have the same beginning fragments as the original database. This information can be obtained from saved database creation scripts, or by running the following command:

```
select segmap,'Size in MB'=size/512 from sysusages where dbid= db_id("database_name")
```

Example:

suppose your database was created with the following statement:

```
create database dbname on datadevice1 = 1000,
```

```
log on Logdevice1 = 200
```

```
go
```

```
alter device dbname on datadevice2 = 500 running:
```

```
select segmap,'Size in MB'=size/512 from sysusages
```

```
where dbid= db_id("dbname")
```

would return:segmap Size in MB

```
3 1000
```

```
4 200
```

```
3 500
```

You could create a 3GB database as follows and load your database into it (using "for load" option will shorten database load time):

```
create database newdatabase on datadevice3 = 1000 log on logdevice3 = 200
```

```
for load
```

```
go
```

```
alter database newdatabase on datadevice 3=500 for load go
```

```
alter database newdatabase on datadevice4=300 for load go
```

```
alter database newdatabase on datadevice5=1000 for load go
```

load database newdatabase from dbname\_dump go

## 4.6 Monitoring Space Usage

### 4.6.1 Thresholds

Thresholds are defined on segments to provide a free space value at which a procedure is executed to provide a warning or to take remedial action.

Use **sp\_addthreshold** to define your own thresholds:

**sp\_addthreshold** database\_name, segment\_name, free\_space, procedure\_name

where free\_space is the number of free pages at which the threshold procedure executes; procedure\_name is the stored procedure which the threshold manager executes when the number of free pages falls below the free\_space value. Please see the section on Auditing later in this chapter for an example of Thresholds.

Example of Threshold Commands mentioned above:

Sp\_addthreshold CustomerDB, "default", 10230, CustDefaultSegWarn

## 4.7 Creating Database Objects

For special cases, creation (and modification) scripts are stored in **\$SYBASE/scripts/scriptname**. There should be a template for each type of object to be created.

### 4.7.1 Example of Creating a User Table

The DBA has received a request to create a new table in the pdps\_db\_ops database called **PGE\_Statistics** which has three column, pge\_id, pge\_statistic\_type, and pge\_statistic.

1. In the **\$SYBASE/scripts/create.db\_objects** directory, DBA creates a script file from the proper template.

```
Syntax: % cd /usr/ecs/OPS/COTS/sybase/scripts/create.db_objects
```

```
        % cp table_template.sql PGE_Statistics_table.sql
```

2. Appropriate items are modified so that the script file resembles the following:

```
1> create table PGE_Statistics (
```

```
2> pge_id                int,
```

```
3> pge_statistic_type int,
```

```

4> pge_statistic          float )
5> go
6> sp_help PGE_Statistic
7> go

```

3. DBA runs the script from the UNIX command prompt:

```

Syntax:      %isql -Usa -Sservername -iPGE_Statistics_table.sql \
             -oPGE_Statistics_table.out

```

4. DBA checks the PGE\_Statistics\_table.out file for success

Other objects are created in like manner but are not included here due to space considerations.

## 4.8 Creating and Managing Logins and Roles

Earlier versions of SQL Server administrative responsibilities needed to be executed by an individual logged in –literally- as sa. Now specific user logins can be assigned components of administrative responsibility, enabling you to track and audit administrative activities.

The three roles are sa\_role (systems administrator) for administrative tasks, sso\_role (site security officer) for security tasks, and oper\_rol (operator) for backup and recovery tasks.

In order to connect to a SQL Server a login must be created by the System Administrator or a system security officer. Login details are stored in the syslogins table in the **master** database.

The system stored procedure **sp\_addlogin** adds new login names to the server but does not grant access to any user database.

```

Syntax: sp_addlogin login_name, password, [,default database ,language, fullname]

```

In order to gain access to a database, the System Administrator, system security officer, or the specific database owner must “add” the user with the **sp\_adduser** system stored procedure.

```

Syntax:      1> sp_adduserlogin_name [ username, group_name]
             2> go

```

### 4.8.1 Example of Creating a Login and Granting Database Access

The DBA has received a request to authorize John Q. Public to the pdps\_db\_ops database.

**\*It is a good practice to have a default\_db, when you create a user account.**

1. In the `$$SYBASE/scripts/create.users` directory, DBA creates a script file containing the `sp_addlogin` command (named `public.sql`)

```
Syntax:      % cd /usr/ecs/OPS/COTS/sybase/scripts/create.users
              % cp template.sql public.sql
```

2. DBA modifies appropriate fields so that the script resembles the following:

```
1> sp_addlogin jpublic,jpublic, default_db
2> go
3> use pdps (OPS mode)      4> go
5> sp_adduser jpublic
6> go
7> sp_helpuser
8> go
```

3. DBA runs the script from the UNIX command prompt:

```
Syntax:      % isql -Usa -Sservername - public.sql -opublic.out
```

4. DBA checks the `public.out` file for success

## 4.9 Permissions

Permissions are used to control access within a database. The DBA uses the **grant** and **revoke** statements to accomplish this. There are two types of permissions within a database, **Object** and **Command**. In general, **Object** privileges control select, insert, update, delete, and execute permissions on tables, views, and stored procedures. **Command** permissions control access to the **create** statements for databases, defaults, procedures, rules, tables, and views.

The syntax for the **grant** and **revoke** statements are quite similar:

```
grant {all [ privileges] | command_list }
to { public | name_list | role_name }
```

```
revoke {all [ privileges] | command_list }
from { public | name_list | role_name }
```

#### 4.9.1 Example of Granting Privileges to a Specific User

The DBA receives a request that John Q. Public should be able to read the DATA\_INFO table and read and update the SUBSCRIPTION\_NOTIFICATION TABLE.

Syntax:           1> **grant** select on DATA\_INFO to jpublic  
                  2> **grant** select, update on SUBSCRIPTION\_NOTIFICATION to jpublic  
                  3> go

Note: It is recommended that the DBA store these command in a “.sql” file in the \$SYBASE/scripts/create.db\_objects directory, along with their results.

## 4.10 Backup and Recovery

**Table 4.10-1. Backup and Recovery Definitions**

Term	Definition
Backup Script Components	Located in the <b>\$SYBASE</b> directory, they include: sybasedump, dmpdb_trns, copy_daily_dumps_to_week1, copy_daily_dumps_to_week2
Backup files	Defined in Table 4.2-2, the location of these files has been determined during server setup
Backup Statements	Generated from the sql in sybasedump these include calls to dbcc, Dump Database, and Dump Transaction commands
Backup Subdirectory	The only directory level underneath of the Backup Directory, defined in Table 4.2-2.
Backup Summary	An extraction of the successful Dump messages along with any errors generated by the Backup Statements stored in the Backup Subdirectory.

### 4.10.1 Automatic Backups

The following are the list of all procedures and scripts files that are currently being used for Sybase backups. There are cron jobs running at all sybase **servers** that have SQL server installed. All dump files are currently written to LOCAL machine. The site DBA is responsible for configuring the backup dump to the REMOTE sybase directory.

To check if the crontab is up and running, enter:

```
> crontab -l
```

- Example of the output:
- 019 \* \* 1-6 /usr/ecs/OPS/CUSTOM/dbms/COM/DBAdmin/EcCoDbSyb\_DumpDb
- 012 \* \* 1-6 /usr/ecs/OPS/CUSTOM/dbms/COM/DBAdmin/EcCoDbSyb\_DumpTran
- 021 \* \* 1-5 /usr/ecs/OPS/CUSTOM/dbms/COM/DBAdmin/EcCoDbSyb\_CkErrorLog

NOTE:

If the crontab is not running enter:

```
> crontab /usr/ecs/OPS/COTS/sybase/run_sybcron
```

The following files will be installed by EcCoAssist to the /usr/ecs/OPS/CUSTOM/dbms/COM/DBAdmin directory:

EcCoDbSyb\_README

EcCoDbSyb\_DumpDb

EcCoDbSyb\_DrumpTran

EcCoDbSyb\_DbStat  
EcCoDbSyb\_SedFile  
EcCoDbSyb\_DboMail  
EcCoDbSyb\_SetupKsh  
EcCoDbSyb\_CkErrorLog  
EcCoDbSyb\_tran\_log.awk

### **SCRIPTS**

### **DESCRIPTIONS**

EcCoDbSyb_SetupKsh	This file contains the SYBASE and DSQUERY (server) environment setup. This file is call by EcCoDbSyb_DumpDb, EcCoDbSyb_DrumpTran, and EcCoDbSyb_CkErrorLog scripts.
EcCoDbSyb_DumpDb	This script contains the code to dump the databases. First, it checks for any DBCC error on the master database, if there is any error on the master, the script sends an email to the DBA and exit the program. If the master database dump was successfull, then the rest of the databases are dumped. Each database has a DBCC check, if there is any error on the database then the database is NOT dumped and an email is send to the DBA. At the end, an status email is send, providing all the database names that were sucefully dumped
EcCoDbSyb_DumpTran	This script contains the code to dump the transaction logs. This dumps the transaction logs for each database, it check the error log file, if the error Msg is 4207 or 4221 it will do a dump of the database firt, then it will do the trasaction dump. If there is any other error Msg then the transaction dump will fail and email will be send. At the end, an status of the transaction log dumps is email to the DBA
EcCoDbSyb_SedFile	This file contains all the database that don't need to be dump (i.e., temp, model, etc.)
EcCoDbSyb_DboMail	This file contains the email list of all the DBA's.
EcCoDbSyb_DbStat	This script updates the index table of a database. This script is called from EcCoDbSyb_DumpDb after each successfully database dump.
EcCoDbSyb_CkErrorLog	This script checks for specific database error messages from the Sybase Error Log File every hour and emails the error messages

to the DBA's in the EcCoDbSyb\_DboMailfile.

EcCoDbSyb\_tran\_log.awk

This script matches the current hour with the hour the error messages were generated in the Error Log File. If errors found, the messages are saved in a mailfile and sent to DBA's.

**THE FOLLOWING FILES MUST BE MODIFIED BEFORE RUNNING ANY OF THE ABOVE SCRIPTS:**

EcCoDbSyb\_SetupKsh

Make user you have the SYBASE files under /usr/ecs/OPS/COTS/sybase

EcCoDbSyb\_SedFile

Add any other database that might not need to be backed up.

The databases that are listed in this file do not need to be backed up.

EcCoDbSyb\_DboMail

Add/delete the email of the DBA and any other email that might need to be added/deleted. All the errors and status will be sent to them.

run\_sybcron

The following is an example on the crontab file that should be run by a sybase user. The first one will run the EcCoDbSyb\_DumpDb script that dumps the databases at midnight from Monday to Saturday.

The second one, EcCoDbSyb\_DumpTran script that dumps the transaction logs will run three times a day, 10AM, 1PM and 4PM from Monday to Saturday. The Third one, EcCoDbSyb\_CkErrorLog that check the SYBASE error log file will run every hour from Monday to Saturday.

```
0 0 * * 1-6 /usr/ecs/OPS/CUSTOM/dbms/COM/DBAdmin/EcCoDbSyb_DumpDb
```

```
0 10,13,16 * * 1-6 /usr/ecs/OPS/CUSTOM/dbms/COM/DBAdmin/EcCoDbSyb_DumpTran
```

```
0 * * * 1-6 /usr/ecs/OPS/CUSTOM/dbms/COM/DBAdmin/EcCoDbSyb_CkErrorLog
```

NOTE: Make sure there is an OPS mode directory with all script files.

All these scripts reside in “/usr/ecs/OPS/CUSTOM/dbms/COM/DBAdmin” directory. The assigned site DBA will be responsible for maintaining, modifying and applying necessary changes that are applicable to their site as for (security, and backup schedule).

SQL Server backups are performed nightly by a **cron** job which runs the **run\_sybcron** program located in the **\$SYBASE/** directory. The following table of definitions will be used throughout the rest of this section.

**Table 4.10-2. Automatic Backup Components**

Component Name	Function(s)
run_sybcron	File added with the crontab -e command, contains several executable cron commands. <b>Example:</b> 00 19 * * 1-6 /data1/COTS/sybase/sybasedump
EcCoDbSyb_DumpDb	Controlling script that performs the following functions: run isql to create the Backup Statements run isql to execute the Backup Statements record the results of the Backup Statements in Backup Files copy the Backup Files to the Backup Subdirectory create the Backup Summary “greps” successful Dump statements along with any errors generated, sends e-mail to the DBA and writes them to the backup_summary file
sp	SQL Server password file - contains password for backup role

No intervention in the Automatic Backup Process is required by the DBA, though periodic checks of the Backup Subdirectories are recommended.

### 4.10.2 Manual Backups

Manual backups can be performed at any time by the System Administrator and are recommended for the following situations:

1. Any change to the **master** database - this includes new logins, devices, and databases
2. Any major change to user databases - a large ingest or deletion of data, definition of indexes
3. Other mission-critical activities - as defined by the DAAC Operations Supervisor.

Both the **dump database** and **dump transaction** command processing are off-loaded to the backup server, and will not affect normal operations of the database. These commands are performed by the System Administrator on appropriate databases as follows:

Syntax:

```
1> dump database master to
"/usr/ecs/OPS/COTS/sybase/sybase_dumps/dumps/dbname.dat.MMDDHHMM."
2> go
```

After dumping the database, compress the dump file by executing:

```
%compress
/usr/ecs/OPS/COTS/sybase/sybase_dumps/dumps/dbname.dat.MMDDHHMM.
```

2> go

### 4.10.3 Manual Recovery

Manual recovery of a user database is performed by the System Administrator by the use of the **load database** and **load transaction** commands. For issues concerning the **master** database, please consult your System Administrator's Guide for assistance. It is recommended that any user database to be recovered be dropped and created with the **for load** option., The **databasename.sql** along with any **alter.databasename.sql** scripts can be , combined into one script which will re-create the user database with the **for load** option. This will insure the success of the **load database** and **load transaction** commands.

### 4.10.4 The BulkCopy Utility

The **bcp** utility is located in the **\$SYBASE/bin** directory and is designed to copy data to and from SQL Server databases to operating system files.

#### 4.10.4.1 Requirements for Using bcp

In general, you must supply the following information for transferring data to and from SQL Server:

- a. Name of the database and table
- b. Name of the operating system file
- c. Direction of the transfer (in or out)

In order to use **bcp**, you must have a SQL Server account and the appropriate permissions on the database tables and operating system files that you will use. To copy data **into** a table, you must have **insert** permission on that table. To copy data **out** to an operating system file, you must have **select** permission on the following tables:

- a. The table being copied
- b. sysobjects
- c. syscolumns
- d. sysindexes

#### **bcp Syntax**

```
bcp [[database_name].owner.]table_name {in | out} datafile [-e errfile] [-n] [-c]
```

[-t field\_terminator] [-r row\_terminator] [-U username] [-S server]

#### 4.10.4.2 Example of User Database Recovery

The database **UserDB** was created using the following script excerpt: (stored in home/scripts/create.databases/userdb.sql)

```
create database UserDB on data_dev1 = 100 log on tx_log1 = 50 [with override]
```

and was modified using the following script excerpt:  
(home/scripts/create.databases/alteruserdb.sql)

```
Alter database UserDB on data_dev1=50
```

For the purposes of this example, the full database backup and transaction log dumps were successful and located in /usr/ecs/OPS/COTS/UserDB.dat and UserDB\_tx.dat

**1.** In the **\$SYBASE/scripts/create.databases** directory, DBA makes a script file from the template.

```
Syntax: % cd /usr/ecs/OPS/COTS/sybase/scripts/create.databases
```

```
% cp template.sql userdb_for_load.sql
```

**2.** Appropriate items are modified so that the script file resembles the following:

```
1> create database UserDB on data_dev2=100 log on tx_log2=50 for load
```

```
2> go
```

```
3> alter database UserDB on data_dev3=50
```

```
4> go
```

**3.** DBA saves the script in **\$SYBASE/scripts/create.databases/userdb\_for\_load.sql**

**4** DBA runs the script from the UNIX command prompt.

```
Syntax: %isql -Usa -Sservername -iuserdb_for_load.sql -ouserdb_for_load.out
```

**5** DBA checks the userdb\_for\_load.out file for success

**6** DBA loads the database from the full backup.

```
Syntax: 1> load database UserDB from
```

```
2> "/usr/ecs/OPS/COTS/sybase/sybase_dumps/week1/dbname.dat.MMDD
```

```
3> go
```

7 DBA loads the transaction file from the transaction file dump.

Syntax: 1> load transaction UserDB from

2> “/usr/ecs/OPS/COTS/sybase/sybase\_dumps/week1/dbname.tran.MMD

3> go

## 4.11 Database Performance and Tuning

Once your application is up and running, the DBA monitors its performance, and may want to customize and fine-tune it. Use the following software tools provided by SQL Server:

- a. Setting query processing options with the **set** command
- b. Setting database options with **sp\_dboption**
- c. Monitoring SQL Server activity with **sp\_monitor**
- d. Using **update statistics** to ensure that SQL Server makes the best use of existing indexes
- e. Changing system variables using **sp\_configure** and the **reconfigure** command
- f. Placing objects on segments to spread i/o, improve throughput, etc. as described in section 4.4.4

For a complete discussion of issues related to SQL Server performance and tuning, refer to your SYBASE SQL Server Performance and Tuning.

## 4.12 Installation of the Applications

DBA should have physical devices configured before installing either autosys or remedy. Both applications use Sybase as their database.

### 4.12.1 Installation of the Application Database

The installation of the application databases has been automated using ECS Assistant. The application databases are created using the DbBuild script which can only be invoked through ECS Assistant or the Command Line. Scripts that ECS Assistant invokes are:

DbBuild - Create new empty database and loads with initial data

DbPatch - Upgrade to new schema while retaining existing data.

### 4.12.2 The AUTOSYS Application and other Configuration Issues

The AUTOSYS application works in tandem with PDPS/DPSs to schedule the jobs that run on Science Processor. Autosys installation is performed in /usr/ecs/OPS/COTS by the auto install program located in the autosys/install directory. The results of the installation are stored in an

autosys\_install.scr file located in the AUTOSYS home directory (/use/ecs/OPS/COTS/autosys). For pdps to run properly with AUTOSYS, the following activities are completed:

- a. A user is defined named **autosys**
- b. **autosys** user is added to the pdps database (OPS mode)
- c. The autosys server is added to the syssservers table with **sp\_addserver**
- d. The server is added to the syssservers table on the AUTOSYS server with **sp\_addserver**

### 4.12.3 Spatial Query Server (SQS)

SQS is a multi-threaded, Sybase Open Query database engine, which is required by the Science Data Subsystem (SDSRV). This product allows definition of spatial data types, spatial operators, and spatial indexing. SQS communicates with Sybase SQL Server to process SDSRV requests to push and pull metadata. SDSRV database server resides on an SGI machine. SQS also, reside on the same machine as SDSRV Sybase SQL Server.

- Named X1acg01 - where X is the DAAC specific identifying character.
- pathname - /usr/ecs/OPS/COTS/sqs222/bin/sqsserver
- Should have one dedicated CPU per instance running. Defaults to one instance now, but may require additional instances later for performance reasons.
- Requires one entry in the Sybase “interfaces” file per instance of the SQS server to be run.
- Consult startup scripts in /etc/init.d/sybase and /etc/init.d/sqs\_222

SQS requires a Sybase login with SA or sa\_role and associated password to start. SQS environment variables requirements:

- SYBASE = Location of the Sybase home directory. Example: /tools/sybOCv(TBD)
- PATH = Must include in this order - /usr/bin; /usr/sbin; \$SYBASE/bin
- DSQUERY = Name of SQL Server to which to connect. From the \$SYBASE/intefaces file. Examples - g0acg01 \_srvr
- DSLISTEN = Name of SQS server to use. Example - g0acg01\_ srvr
- SQSUSER = Name of the user (SA or sa\_role) for system connection.
- SQSPASSWORD = Password for the system connection login

The SQS startup script requires the following information:

SQSHOME = location of sqsserver binaries.

The following is a list of options that can be imbedded in the startup script, these options are beneficial, but they are not required.

#### 4.12.3.1 SQS STARTUP OPTIONS:

- -e path of the SQS server logfile. Example /usr/ecs/OPS/COTS/sqs222/sqs/bin/sqs\_222.log
- -u number of concurrent SQS connections. Recommend minimum of 125. Example -u 125
- Usually started with a delay, after the SQL Server is started. This delay be sufficient for the SQL server to recover and come-up.
- \$SQSHOME/bin/sqsserver -e \$SQSHOME/sqs\_222.log -u \$USER &

SQS has dependencies on Sybase, such as:

- Sybase must be running prior to starting SQS
- SQS user id that starts SQS, which is different from the application user ID must have admin privileges
- SQS opens a connection to Sybase's because it writes to the Sybase System tables
- SQS server thread runs under the userid sa. In order to avoid confusion when monitoring this thread, it is best to:
  - create a separate login and userid specifically to monitor SQS
  - grant sa\_role authority to the userid created to monitor SQS

EXAMPLE: 1> **sp\_adduser** sqs\_mon

2> **grant** sa\_role to sqs\_mon

3> go

### 4.13 Passwords Security

Security has become a sensitive issue throughout the IT Industry. The ECS program is also concern about security and the risks associated with security. As a result the following directive is issued to all DAACs.

All System Administrators and Database Administrators at the sites are responsible for easonable security measures when installing ECS custom software. This means:

1. Changing the permissions of online secure files to the minimum level required .
2. Backing up secure file(s) to removable media (floppy or tape) and removal of secure files immediately after installation is complete.
3. The media should then be kept in a secure location.

**The following file is affect as result of this requirement on the ECS program.**

A. /usr/ecs/<MODE>/CUSTOM/dbms/<SUBSYSTEM>/Ec<server>SybaseLogins.sql

B. Set permissions to 711 (user read, write, execute, group and other read only)

Figures 4.X-1 and 4.x.-2 are the Technical Directives issued by the Director of Systems Engineering.

Raytheon  
Systems  
Engineering Technical Directive DRAFT  
ECS Project  
No. 98-XXX  
Subject: Flp account password distribution  
The following directive is issued to all DAACS and labs.  
January 15, 1999

**Issue:** Flp accounts are required for ECS custom software to function. Accounts and passwords have been distributed together via public means (such as electronic mail) that presents an unacceptable security risk.

**Fix:** When communicating sensitive information such as accounts and passwords, do not put them together in the same document.

**Testing:** N/A

**Implementation:** For distribution of account information when it is not hand carried:

1. Send the account information in one document.
2. Send the password information in a different document in inverted order (the first password entry is matched to the last account entry).

NOTE: It is not required but it is recommended to encrypt flp account information using the UNIX crypt command when sending via electronic means. If crypt is used, the password used should be contained in a third document.

**Point of Contact:**

**Approved By:** M. McBride  
Director, Systems Engineering

**Reference CCR:** 98-XXXX  
-----End of Directive-----

Figure 4.x-1 Technical Directive

The following directive is issued to all DAACS and labs.

**Issue:** System verification testing has revealed that passwords are being written in clear text to log files.

**Implementation:**

1. This practice presents an unacceptable security risk to ECS.
2. Names of system accounts may be written to logs. However, logs that are so written must have file permissions on the log file changed to 770 with the cm<MODE> account being the owner and cm<MODE> group being the group. If logs are retained on the host, they must be encrypted using the UNIX crypt command and a locally agreed upon password. One such password per location is acceptable but the password may not be kept in clear text on any system.

**Point of Contact:** Byron Peters, Chair, TSIWG, email - [bpeters@ecos.kbr.com](mailto:bpeters@ecos.kbr.com),  
phone 301/883-4077

**Approved By:** M. McBride  
Director, Systems Engineering

**Reference CCR:** 98-XXXX  
-----End of Directive-----

Figure 4.x-2 Technical Directive