BENOHEAD'S SYBASE ASE COOKBOOK

Contents

Introduction

I've compiled in this short ebook all posts I've ever written regarding Sybase ASE and tried to cluster them a little bit. This ebook is full of information useful when you work daily or once in a while with Sybase ASE. Sybase ASE is a high-performance and scalable database management system but it is also complex and not always easy to understand.

After working for over 10 years with ASE, I've gathered a lot of information I share on a daily basis with colleagues and once in a while on my blog at http://benohead.com.

I provide this ebook in the hope to be helpful. It is thus available for free. Since I'm not a professional writer, I do not have a crew of people proof-reading it. So spelling might be not as good as it should be and I can't be sure that everything in there is 100% accurate. If you find any mistake, please contact me at henri.benoit@gmail.com and I'll do my best to update it.

The transaction log

Transaction log full during recovery of dbccdb

On one of our test server, we had an abrupt reboot while ASE was rolling back a huge transaction. After restart, Sybase recovered that database and then did a dbccdb recovery as well. During the ANALYSIS and REDO passes went fine but during the UNDO pass, the transaction log got full:
Can't allocate space for object 'syslogs' in database 'dbccdb' because 'logsegment' segment is full/has no free extents. If you ran out of space in syslogs, dump the transaction log. Otherwise, use ALTER DATABASE to increase the size of the segment.

Dumping the transaction log did not work since the database was recovering and couldn't be accessed. So we had to use a more hardcore way known as log suicide i.e. short-circuiting the recovery. This is generally not recommended and I wouldn't have done it on our database but on dbccdb, it was worth a try.

First you need to allow updates on system tables as we need to manipulate the sysdatabases table:

```
use master
go
sp_configure "allow updates",1
go
```

The we'll update the status of dbccdb:

```
update sysdatabases set status=-32768, status2=0 where name='dbccdb'
```

status=-32768 means bypass recovery mode.
status2=0 clears the info about the currently running recovery.

Then shutdown ASE:

```
shutdown with nowait
```

Now you can manually start the dataserver. When the dataserver is up again, execute the following to truncate the transaction log of dbccdb:

```
dump tran dbccdb with no_log
```

And set back the status of dbccdb to the default value:

```
update sysdatabases set status=4, status2=-32768 where name='dbccdb'
```

status=4 means that select into/bulkcopy are allowed.
status2=-32768 means that dbccdb has some portion of the log which is not on a log-only device. if it is not the case for you, use 0 instead.

Extending the transaction log

This will show you how to extend the transaction log on a user database by adding an additional log device. In many cases dumping the transaction log or increasing the size of the existing device would be better but in some occasions, I've needed this...

First you have to create a new device on the hard disk using “disk init”:

```
use master
go
disk init name = "log_2_dev", physname = "/db_data/devices/log_2_dev",
size=204800
go
```

In this example, “log_2_dev” is the name of the new device in ASE. It will be created in the file "/db_data/devices/log_2_dev". And its size will be 400 MB (204800 pages, 2KB each). Instead of specifying a size in 2K-blocks you can also write 400M to directly define a size in megabytes.

You can check whether the device has been properly created like this:

```
sp_helpdevice
go
```

Then you need to assign this device to the transaction log of your database (assuming the name of the database is mydb):

```
use master
go
```

```
alter database mydb log on log_2_dev = '400M'
go
```

It tells Sybase to additionally use the new device for the transaction log (using 400 MB).

You can check with the following whether everything went fine:

```
sp_helpdb mydb
go
```

You should see two log devices e.g.:

```
1> sp_helpdb mydb
2> go
 name                     db_size       owner                    dbid
         created
         status
 ------------------------ ------------- ------------------------ ------
         --------------
         --------------------------------------------------------------------
----------------------------------
 mydb                         1797.0 MB sa                            4
         Jan 24, 2012
         select into/bulkcopy/pllsort

(1 row affected)
 device_fragments               size          usage
         created             free kbytes
 ------------------------------ ------------- --------------------
         ------------------- ----------------
 data_1_dev                          991.0 MB data only
         Jan 24 2012 11:33PM           809514
 log_1_dev                           109.0 MB log only
         Jan 24 2012 11:33PM not applicable
 index_1_dev                         297.0 MB data only
         Jan 24 2012 11:34PM           243404
log_2_dev                            400.0 MB log only
         Jan 25 2012  3:57AM not applicable

 ---------------------------------------------------------------
 log only free kbytes = 292174
(return status = 0)
```

After that you shouldn’t have problems anymore with the transaction log getting full too quickly.

If you need to remove the log device later on, please use the following:

```
use master
go
sp_dboption mydb, "single user", true
go
use mydb
go
sp_dropsegment 'logsegment','mydb','log_2_dev'
```

```
go
use master
go
sp_configure 'allow updates', 1
go
delete from sysusages where dbid=db_id('mydb') and vstart=(select low from
sysdevices where name='log_2_dev')
go
sp_configure 'allow updates', 0
go
sp_dropdevice 'log_2_dev'
go
sp_dboption mydb, "single user", false
go
```

The sing user mode means that no other connection than the one you're working with will be available. This ensures that no other activities are done in parallel.
Allow updates allows you to modify system tables manually.

Check for which objects you have transactions in the log

If you see that your transaction log is growing at an unexpected speed or are just curious which transactions are currently contained in the log, you can use the dbcc log operation:

```
dbcc traceon(3604)
go
dbcc log(mydbname, 0)
go
```

Replacing mydbname by the name of the database which transaction log you want to inspect. This will return quite some data. If you only want to see which objects (e.g. tables) are affected, you can use the following shell command:

```
cat <<EOT | isql -Usa -Pmysapassword | grep "objid=" | awk ' { print $1; }' |
awk -F "=" '{ print $2; }' | sort | uniq -c | sort -n -r
dbcc traceon(3604)
go
dbcc log(mydbname, 0)
go
EOT
```

If isql is not in your PATH, write the full path instead (probably $SYBASE/OCS/bin/isql on a Linux box).

This will get the log data, extract the lines containing the object ID, then extract the object IDs, counting the number of time each one appears and sorting the list by decreasing number of occurences.

On my machine it returns something like this:

```
     10 2062679415
```

```
      6 20
      1 8
```

2062679415 being a test table I've filled for the purpose of this test.
8 is syslogs.
20 is sysanchors.

You can check what those returned object IDs are with the following statement run in your database:

```
1> select id, name from sysobjects where id in (8,20,2062679415)
2> go
 id          name

 ----------- --------
           8 syslogs
  2062679415 henri
```

Note that sysanchors (object ID 20) is not shown here because it is not a normal database table. It's a per-database pseudo-catalog where row anchors used by system catalogs are stored.

Also keep in mind that checking the contents of the transaction log takes longer when it's already full. So if you see that the log is filling at a constant rate, it might be a good idea to read from the log not too long after the last dump database or dump of the transaction log, so that it is faster (it will have an engine use 100% CPU for quite some time if it's already full).

Disable the transaction log

In some cases, you keep getting problems with a transaction log always filling up but do not need to be able to restore all data in a disaster recovery scenario (e.g. because you're initially filling the database and if something goes wrong, you can just repeat the process or because it's a test or development system and you do not care about the data).

Unfortunately, it is not possible to completely disable it. But you can make sure that the transaction log will be truncate on every checkpoint. The transaction log is then still there. It still costs you resources but it will be cleared at every checkpoint which will prevent it from filling up.

In order to have it truncated on checkpoint, use the following command:

```
exec master..sp_dboption mydb, 'trunc log on chkpt', true
```

Replace mydb by the name of the database on which you want to perform it.

To reenable the transaction log, just execute the following:

```
exec master..sp_dboption mydb, 'trunc log on chkpt', false
```

Transaction log dump and non-logged operations

Most of you probably already know this but just to make sure... There are three kinds of operations from a transaction log perspective:

1. logged operations: those are operations which are completely logged in the transaction log.
2. minimally logged or non-logged operations which are logged in the transaction log but not every change performed by this operation is logged. They do not prevent a dump and restore of the transaction logged
3. minimally logged or non-logged operations which are not logged in the transaction log at all.

Here are a few example of each type of operations:

1. insert into, update, delete
2. truncate table
3. select into, fast bcp, parallel sort

When an operation of the third category is performed, since the transaction log entries are missing, a dump and restore of the transaction log only is not possible anymore. This means ASE is not able to recover the database in a disaster recovery scenario unless you take a full dump of the database. Since the dumped transaction log does not contain the required information, ASE prevents you from dumping the transaction log once one of these operations has been performed because you couldn't use the dumped log to recover the database anyway. Many people tend to think that truncate table also prevents a transaction log dump, which is not the true. Truncate table does not log every deletion in the table and is thus not a fully logged operation but it does log all page deallocations in the transaction log so that it's still possible to reconstruct the database. So if you rely on a transaction log dump to recover the database or if you use it to migrated data from a system to the other one, it is important to:

- prevent such operations to happen
- check whether such operations have been performed

prevent such operations to happen This can be done by setting an option on the database:

```
master..sp_dboption mydb, "select into/bulkcopy/pllsort", false
go
use mydb
go
checkpoint
go
```

Replace mydb by the name of the database you want to prevent such operations on. With this option set select into, fast bcp and parallel sort operations will not be allowed anymore on this database. **check whether such operations have been performed** You can use the following query if such operations are not prevented as shown above.

```
select tran_dumpable_status('mydb')
```

If it returns 0, then everything is fine and a dump of the transaction log should work fine. Otherwise, such an operation was performed and it is not possible to dump the transaction log until a full dump of the database is performed.

If the returned value is not 0, you can find out exactly what happens by interpreting the return bit mask:

- 1 – the specified database name was not found.
- 2 – there is no separate log device for this database.
- 4 – the log first page is on a data-only disk fragment.
- 8 – the **trunc log on chkpt** option is set for the database.
- 16 – an operation preventing a dump of the transaction log was performed.
- 32 – a transaction log dump with truncate_only was performed.
- 64 – it's a new database and a full database dump must be performed before you can dump the transaction log.

Here are a few explanations on the different values:

- 1 - this mostly means that you've misspelled the database name.
- 2 or 4 - the transaction log needs to be entirely on a separate transaction log device for the dump of the transaction log to work.
- 8 - the database is configured to truncate the transaction log on checkpoint. This means that the transaction log is mostly empty except for things which happened since the last checkpoint. A dump of the transaction log then makes no sense, since most of the changes performed have already been discarded. See Disable the transaction log for more information.
- 16 - a select into, fast bcp or parallel sort has been performed and the transaction log would be useless anyway.
- 32 - the transaction log has been truncated by this command and the chain has been broken.
- 64 - the transaction log contains the changes applied since the last full dump so you need at least one full dump in order to use it.

So if you do not need any of the operations which prevent the transaction log from being dumped, the best solution is to prevent them on the database level. Otherwise, when you need to perform a transaction log dump (e.g. because the transaction is full or because you need to migrate the changes to a replicated database), you should first check whether a transaction log dump would be possible and trigger a full dump instead if not.

Database dump

Get information about a dumped database

In ASE 15, it is possible to get information from a dump file directly using the load database command with the "with headeronly" parameter:

```
1> load database master from '/db/dump1/master.201209170128' with headeronly
2> go
Backup Server session id is: 69. Use this value when executing the
'sp_volchanged' system stored procedure after fulfilling any volume change
request from the Backup Server.
Backup Server: 6.28.1.1: Dumpfile name 'master12261014D2 ' section number 1
mounted on disk file '/db/dump1/master.201209170128'
This is a database dump of database ID 1, name 'master', from Sep 17 2012
1:28AM. ASE version: Adaptive Server Enterprise/15.5/EBF 18164 SMP
ESD#2/P/x86_64/Enterprise Linux/asear155/2514/64-bit/F. Backup Server
version: Backup Server/15.5/EBF 18164 ESD#2/P/Linux AMD Opteron/Enterprise
Linux/asear155/3197/64-bit/OPT/We. Database page size is 4096.
Database contains 15360 pages; checkpoint RID=(Rid pageid = 0x30e5; row num =
0x20); next object ID=1929054877; sort order ID=50, status=0; charset ID=1.
Database log version=7; database upgrade version=35; database
durability=UNDEFINED.
segmap: 0x00000007 lstart=0 vstart=[vpgdevno=0 vpvpn=4] lsize=6656
unrsvd=3461
segmap: 0x00000007 lstart=6656 vstart=[vpgdevno=0 vpvpn=22532] lsize=8704
unrsvd=8223
```

This will not really load the dump but just display information about the dump. In the example above, you can see the following:

- which database was dumped: ID/name: 1/master
- when was the dump performed: Sep 17 2012 1:28AM
- which ASE version is running on the server where the dump was created: 15.5
- the page size on that server: 4K
- number of pages, next object ID, sort order, status, charset
- the segmentation map

Dump load failed

We've had the following issue: After a problem on a customer database, we needed to restore a single table. A compressed dump was available containing the latest version of this one table but an older version of the other tables. So the plan was to load the dump on a test system with the same device layout, ASE version and operating system, export the table and import it on the live system.

Unfortunately while loading the dump, we got the following error:

```
Oct  7 10:57:00 2013: Backup Server: 2.23.1.1: Connection from Server SYBASE
on Host myhost with HostProcid 7705.
Oct  7 10:57:00 2013: Backup Server: 4.132.1.1: Attempting to open byte
stream device: 'compress::9::/dump1_1/mydb.201310030015.000::00'
Oct  7 10:57:00 2013: Backup Server: 6.28.1.1: Dumpfile name 'mydb13276003C2
' section number 1 mounted on byte stream
'compress::9::/dump1_1/mydb.201310030015.000::00'
Oct  7 10:57:23 2013: Backup Server: 4.188.1.1: Database mydb: 2310872
kilobytes (1%) LOADED.
Oct  7 10:57:32 2013: Backup Server: 4.124.2.1: Archive API error for
device='compress::9::/dump1_1/mydb.201310030015.000::00': Vendor application
```

```
name=Compress API, Library version=1, API routine=syb_read(),
Message=syb_read: gzread() error=-1 msg=1075401822
Oct  7 10:57:32 2013: Backup Server: 6.32.2.3:
compress::9::/dump1_1/mydb.201310030015.000::00: volume not valid or not
requested (server: n byte stream 'cu□
@ess::9::/dump1_1/mydb.20¤D, session id: 17.)
Oct  7 10:57:32 2013: Backup Server: 1.14.2.4: Unrecoverable I/O or volume
error.  This DUMP or LOAD session must exit.
```

So it looks like there was a problem uncompressing the dump. I am not too sure where the strange characters in the second to last line come from but I'm not sure either that it's related to the problem.

Reading the header from the dump as described in a previous post worked fine. So the dump was not completely corrupt. It's also the reason why the first percent of the dump could be loaded.

We also tried loading the dump using the "with listonly" option but it failed:

```
1> load database mydb from "compress::9::/dump1_1/mydb.201310030015.000" with
listonly=full
2> go
Backup Server session id is:  26.  Use this value when executing the
'sp_volchanged' system stored procedure after fulfilling any volume change
request from the Backup Server.
Backup Server: 4.22.1.1: Option LISTONLY is not valid for device
'compress::9::/dump1_1/mydb.201310030015.000::00'.
```

I never found out why it wasn't possible to use listonly on this dump file but I didn't really have time to look into it in details...

The I saw that there was a with verify only option. Here from the Sybase documentation:

load database database_name
from [compression=]stripe_device
[at backup_server_name]
[density = density_value,
blocksize = number_bytes,
dumpvolume = volume_name,
file = file_name]
with verify only [= header | full]

But it failed saying there was an error near "only"... Then I wondered why the syntax would be "with headeronly" and "with listonly" but "with verify only" i.e. with an extra space. So we tried without the space and it worked. Well, kind of... It could still load the header but failed with the same error message while reading the rest.

Next I thought it might have been a problem while transferring the dump through FTP (I wasn't sure whether the transfer was done in binary or ASCII mode). One way to check it is to search for \r\n characters in the dump. It can be done using the od command. od dumps files in octal and other formats. You can use the -c option to show special characters as escape characters (i.e. \r and \n). So you need to run od and pipe it to a grep e.g.:

```
od -c mydb.201310060030.000 | grep "\\r  \\n"
```

If this returns something then you probably used the wrong transfer mode using FTP. It wasn't the case here. Also both the files on-site and the one we got had the same number of bytes which also indicates that it's not such an issue.

Another thing you need to check is whether uncompressing failed because of memory or disk space issues. In our case we had plenty of free disk space and RAM available.

Another thing I found while googling for a solution was the following in a newsgroup:

Backup Server will use asynchronous I/O by default and there was a CR 335852 to work around this behavior. Try starting backupserver using trace flag -D32 .

CR Description :-
6.21 Dumping or loading databases with asynchronous I/O
[CR #335852] On an IA32 running Red Hat, running a dump or load database command can cause Backup Server to stop responding when using asynchronous I/O. Backup Server uses asynchronous I/O by default.

[Workaround : Start Backup Server using trace flag -D32 to force a synchronous I/O.

So we tried adding the flag to the start script of the backup server. But it didn't help. Anyway we didn't know whether the problem was during loading or whether there had been a problem while dumping.

The next thing which came up to my mind was to try and uncompress the dump file manually to see whether it's corrupt. This can be done with gunzip. You just need to rename the file in case it doesn't have a valid gzip extension e.g.:

```
mv mydb.201310060030.000.gz
gunzip mydb.201310060030.000.gz
```

In our case it failed. So we repeated it on a dump file we knew was fine and it worked. So we had the source of the problem. The dump stripe was corrupt.
Repeating it on the dump on site worked. So the stripe was not corrupt after the dump but was somehow corrupted in the transfer. So all we had to do was to transfer it again.

I'm not too sure why the stripe got corrupted during the transfer but was happy it didn't get corrupted while dumping as we had feared in the beginning.

Using archive databases

Archive databases are used to access data from a backup file directly without having the restore the database. Let's say you lost some data in a table but had many other changes to other tables since the last backup. Just loading the last backup is not an option since you'd lose everything since the last backup. Of course, if you work with transaction log dumps, you can reduce the loss of data but very often it's still too much. Additionally, in some cases you know the data you want to reload have not changed since the last backup (i.e. some kind of master data). So the best solution would be to be able to keep the current database but just reload this one table. Or maybe you do not want to reload a complete table but just copy a few deleted lines back in a table.

That's exactly what an archive database is for. You cannot dump an archive database. An archive database is just a normal database dump loaded in a special way so that you can access the data without having to do a regular load of the dump which would overwrite everything.

So what do you need in order to mount a database as an archive database. Well, you need two additional databases:

1. A "scratch database"
2. An archive database

The "scratch database" is a small database you need to store a system table called sysaltusages. This table maps the database dump files you are loading to the archive database.

The archive database is an additional database you need to store "modified pages". Modified pages are pages which are created additionally to the pages stored in the dump files. These are e.g. the result of a recovery performed after loading the database dump. So this database is typically much smaller than the dump files you are loading. But it is difficult to tell upfront how big it will be.

So once you have loaded an archive database, the data you see come from these three sources:

- The loaded dump files
- The scratch database
- The archive database

So let's first create the two database (I assume here you have some devices available to create these databases).

First we need to create the scratch database:

```
use master
go
create database scratchdb on scratch_data_dev='100M' log on
scratch_log_dev='100M'
go
```

This will create the scratch database and take in online. Then we need to mark this database as a scratch database:

```
sp_dboption 'scratch', 'scratch database', 'true'
go
use scratch
go
checkpoint
go
```

Then we need to create the archive database:

```
use master
go
create archive database archivedb on archive_data_dev='100M' with
scratch_database = scratchdb
go
```

Replace scratch_data_dev, scratch_log_dev and archive_data_dev by the names of the devices you want to create the data and log fragments of the scratch database and the data fragment of the archive database.

Now we're ready to load the dump. Just do it the way you would load the database to restore it but only load it to the just created archive database e.g.:

```
load database archivedb from '...'
```

You can of course also load the transaction logs with:

```
load transaction archivedb from '...'
```

Note that while loading the database dump or the transaction log dumps, you might get error message saying that either the transaction log of the scratch database or the modified pages section of the archive database run full e.g.:

There is no more space in the modified pages section for the archive database 'pdir_archive_db'. Use the ALTER DATABASE command to increase the amount of space available to the database.

Depending on the message you get, you'll have to add more space for the transaction log of the scratch database or extend the archive database using alter database. Note that ASE usually gives you a chance to do it before aborting. But at some point in time, it will probably abort, so do not take your time ;-)

If you do not care about the recovery and have limited storage available for the archive database you can use:

```
load database archivedb from '...' with norecovery
```

Loading with norecovery also reduces the time required to load. Also the database is automatically brought online (this also means you cannot load additional transaction logs). The downside is that the database might be inconsistent (from a physical and transactional point of view).

If you did not use the norecovery option, you have to bring the archive database online:

```
online database archivedb
```

Once you are done with the archive database and do not need it anymore, you can just drop both databases:

```
drop database archivedb
drop database scratchdb
```

Caches

Force loading in cache

You can load an entire index into Sybase's cache by executing the following:

```
select count(*) from <tablename> (index <indexname> lru)
```

index <indexname> forces the optimizer to use the specified index.
lru tells Sybase to use the LRU Buffer Replacement Strategy i.e. keep the data in cache after the execution of the statement.

This will perform an index scan and load the index data in cache.

If you want to perform a table scan instead i.e. load the data pages in cache, you can write the table name instead of the index name. But if you have an unclustered index with the same name as the table, it will be loaded in cache instead to make sure that the table is loaded, you can issue the following statement:

```
select count(*) from <tablename> (index 0 lru)
```

Please note that this will not load the text and image columns of the table.

In order to check whether the whole table is in cache, you can do the following:

```
select DBName+".."+ObjectName, CachedKB*100/TotalSizeKB as percentage from
monCachedObject where IndexID = 0 order by percentage desc
```

This lists all tables in cache and the percentage of the data pages loaded.

Check contents of the procedure cache

In order to peek into the procedure cache, you can use the following dbcc command:

```
dbcc procbuf
```

In order to see the output on the console, use:

```
dbcc traceon(3604)
go
dbcc procbuf
go
```

You'll see that the output is pretty extensive. If what you are after is which trigger and procedures are using space in the procedure cache and how much space it uses, you only are interested in the lines like:

```
...
    Total # of bytes used               : 1266320
...
pbname='sp_aux_getsize'   pbprocnum=1
...
```

You can thus execute it and grep for these two lines:

```
$SYBASE/OCS/bin/isql -Usa -Pxxxx << EOT | grep "pbname
Total # of bytes used"
dbcc traceon(3604)
go
dbcc procbuf
go
EOT
```

You of course need to replace xxxx by your actual password.

Then you will want to make it looks nicer:

- Merge the two lines: awk '!(NR%2){print p" "$0}{p=$0}'
- Display only the name and the size: awk '{ print $1" "$9" bytes"; }' | sed "s/pbname=//g" | sed "s/'//g"
- Sort by size: sed "s/'//g" | sort -k2 -n

Putting it all together:

```
$SYBASE/OCS/bin/isql -Usa -Pxxxx << EOT | grep "pbname
Total # of bytes used" | awk '!(NR%2){print p" "$0}{p=$0}' | awk '{ print $1"
"$9" bytes"; }' | sed "s/pbname=//g" | sed "s/'//g" | sort -k2 -n
dbcc traceon(3604)
go
dbcc procbuf
go
EOT
```

You will then see something like:

```
...
sp_jdbc_tables 62880 bytes
sp_getmessage 68668 bytes
sp_aux_getsize 80596 bytes
```

```
sp_mda 81433 bytes
sp_mda 81433 bytes
sp_drv_column_default 90144 bytes
sp_dbcc_run_deletehistory 133993 bytes
sp_lock 180467 bytes
sp_helpsegment 181499 bytes
sp_dbcc_run_summaryreport 207470 bytes
sp_modifystats 315854 bytes
sp_autoformat 339825 bytes
sp_spaceused 353572 bytes
sp_jdbc_columns 380403 bytes
sp_do_poolconfig 491584 bytes
sp_configure 823283 bytes
```

Identity columns

Using identity_insert to insert data in tables with an identity column

I have a stored procedure working which get the name of a table as input and does some processing on the table. It first copies some rows from this table to a tempdb table using select into. It then inserts some additional rows based on some logic. The stored procedure is written in a generic way and doesn't know upfront on which table it will be working. Some of the tables have an identity column. So after the select into, the tempdb table also has an identity column. The second statement (an insert into) then fails if there is an identity column as it is by default not possible to specify the value for an identity column.

In order to be able to do it, you need to set the identity_insert option to on:

```
set identity_insert tempdb..mytemptable on
go
```

But if you set it for a table without any identity column, you get the following error message:

Cannot use 'SET IDENTITY_INSERT' for table 'tempdb..mytemptable' because the table does not have the identity property.

So you basically first need to check whether an identity column exists for this table and only set the option for this table if it does. This can be done with the following statements:

```
if exists (select 1 from tempdb..sysobjects o, tempdb..syscolumns c where
o.id=c.id and c.status&128=128 and o.name='mytemptable')
begin
        set identity_insert tempdb..mytemptable on
end
```

(Replace tempdb by the database containing your table and mytemptable by the name of your table)

This first checks whether there is any column for this table which has the status bit 128 set (i.e. is an identity column) and only then sets identity_insert to on for this table.

Another way to do it is:

```
if next_identity('tempdb..mytemptable') is not null
begin
        set identity_insert tempdb..mytemptable on
end
```

next_identity return null if there is no identity column in the specified column.

Additionally you should note that identity_insert can only be set to on for a single table in a given database at a time (within one session). If it is already on for a table and you try to set it on for another one, you'll get the following error message:

Unable to 'SET IDENTITY_INSERT' for table 'tempdb..mytemptable' because IDENTITY_INSERT or IDENTITY_UPDATE is already ON for the table 'myothertemptable' in database 'tempdb'.

So you first have to set it off for the other table:

```
set identity_insert tempdb..myothertemptable off
go
set identity_insert tempdb..mytemptable on
go
```

Of course if you're doing this all programatically it doesn't help you much. So you'd first need to find out whether it's already on for another table. Unfortunately there seems to be no way to get this kind of information from ASE. The only solution to this problem seems to loop through all tables in the current database having an identity column and set identity_insert to off for this table. This can be done using the following stored procedure:

```
if exists (select 1 from sysobjects where name = 'sp__clearidentityinsert')
        drop procedure sp__clearidentityinsert
go

create procedure sp__clearidentityinsert
as
begin
        declare @table_name varchar(255)
        declare @sqlstatement varchar(512)

        declare id_ins_cursor cursor
        for select distinct o.name from sysobjects o, syscolumns c where
o.id=c.id and c.status&128=128
        for read only

        open id_ins_cursor
        fetch id_ins_cursor into @table_name
        while @@sqlstatus = 0
        begin
                set @sqlstatement = 'set identity_insert '+@table_name+' off'
                exec( @sqlstatement )
                fetch id_ins_cursor into @table_name
        end

        close id_ins_cursor
        deallocate cursor id_ins_cursor
end
go
```

Just execute it without parameters and identity_insert will be set to off for all tables in the current database. The following should now work without problem:

```
use tempdb
go
set identity_insert myothertemptable on
go
sp__clearidentityinsert
go
set identity_insert mytemptable on
go
```

If you want to set it for tables in another database than the current database, you'll need to hardcode the database name in the stored procedure before sysobjects, syscolumns and in @sqlstatement e.g.:

```
if exists (select 1 from sysobjects where name =
'sp__clearidentityinserttempdb')
        drop procedure sp__clearidentityinserttempdb
go

create procedure sp__clearidentityinserttempdb
as
begin
        declare @table_name varchar(255)
        declare @sqlstatement varchar(512)

        declare id_ins_cursor cursor
        for select distinct o.name from tempdb..sysobjects o,
tempdb..syscolumns c where o.id=c.id and c.status&128=128
        for read only

        open id_ins_cursor
        fetch id_ins_cursor into @table_name

        while @@sqlstatus = 0
        begin
                set @sqlstatement = 'set identity_insert
tempdb..'+@table_name+' off'
                exec( @sqlstatement )
                fetch id_ins_cursor into @table_name
        end
        close id_ins_cursor
        deallocate cursor id_ins_cursor
end
go
```

Parametrizing the database name is not so easy since cannot use a parameter for the database name in the select of the cursor. We need to use some tricks using a temporary table and dynamic SQL:

```
if exists (select 1 from sysobjects where name = 'sp__clearidentityinsert')
        drop procedure sp__clearidentityinsert
go

create procedure sp__clearidentityinsert(@dbname varchar(255) = '')
as
begin
        declare @table_name varchar(255)
        declare @sqlstatement varchar(1024)

        if @dbname = '' select @dbname = db_name()

        if exists (select 1 from tempdb..sysobjects where type='U' and
name='clearidentityinsert')
        begin
                drop table tempdb..clearidentityinsert
        end

        create table tempdb..clearidentityinsert(name longsysname not null)

        set @sqlstatement = 'insert into tempdb..clearidentityinsert select
distinct o.name from '+@dbname+'..sysobjects o, '+@dbname+'..syscolumns c
where o.id=c.id and c.status&128=128'
        print @sqlstatement
        exec( @sqlstatement )

        declare id_ins_cursor cursor for select name from
tempdb..clearidentityinsert for read only

        open id_ins_cursor
        fetch id_ins_cursor into @table_name

        while @@sqlstatus = 0
        begin
                set @sqlstatement = 'set identity_insert
'+@dbname+'..'+@table_name+' off'
                print @sqlstatement
                exec( @sqlstatement )
                fetch id_ins_cursor into @table_name
        end

        close id_ins_cursor
        deallocate cursor id_ins_cursor

        drop table tempdb..clearidentityinsert
end
go
```

If you call sp__clearidentityinsert without a parameter, it will run on the current database. Otherwise it will run on the specified database e.g.:

```
sp__clearidentityinsert tempdb
```

Copy a table with SELECT INTO ignoring the IDENTITY property

When you use SELECT INTO to copy data from one table to a new table (e.g. to create a copy of the table) and your source table has an identity column, the corresponding column in the newly created table will also be an identity column. Here an example:

We have a table with an indentity column:

```
1> create table mytable_with_id (
2>  id int identity,
3>  something char(1))
4> go
1> sp_help mytable_with_id
2> go
...
 Column_name Type Length Prec Scale Nulls Default_name Rule_name
Access_Rule_name Computed_Column_object Identity
 ----------- ---- ------ ---- ----- ----- ------------ --------- ------------
---- ---------------------- ----------
 id          int       4 NULL  NULL     0 NULL         NULL      NULL
NULL                            1
 something   char      1 NULL  NULL     0 NULL         NULL      NULL
NULL                            0
...
```

We select into a new table:

```
1> select * into mycopy from mytable_with_id
2> go
```

And get a table with an identity column:

```
1> sp_help mycopy
2> go
...
 Column_name Type Length Prec Scale Nulls Default_name Rule_name
Access_Rule_name Computed_Column_object Identity
 ----------- ---- ------ ---- ----- ----- ------------ --------- ------------
---- ---------------------- ----------
 id          int       4 NULL  NULL     0 NULL         NULL      NULL
NULL                            1
 something   char      1 NULL  NULL     0 NULL         NULL      NULL
NULL                            0
...
```

One workaround to this is to convert the id column to an int. It then retains the same value but the column is not an identity column anymore:

```
1> select convert(int, id) as id, something into mycopy from mytable_with_id
2> go
1> sp_help mycopy
2> go
...
 Column_name Type Length Prec Scale Nulls Default_name Rule_name
Access_Rule_name Computed_Column_object Identity
 ----------- ---- ------ ---- ----- ----- ------------ --------- ------------
---- ---------------------- ----------
 id          int       4 NULL  NULL     0 NULL         NULL      NULL
NULL                            0
 something   char      1 NULL  NULL     0 NULL         NULL      NULL
NULL                            0
...
```

Since the identity property is also not propagated when using a UNION, you can also do the following:

```
1> select id, something into mycopy from mytable_with_id UNION select id,
something from mytable_with_id where 1=0
2> go
1> sp_help mycopy
2> go
...
 Column_name Type Length Prec Scale Nulls Default_name Rule_name
Access_Rule_name Computed_Column_object Identity
 ----------- ---- ------ ---- ----- ----- ------------ --------- ------------
---- ---------------------- ----------
 id          int       4 NULL  NULL     0 NULL         NULL      NULL
NULL                            0
 something   char      1 NULL  NULL     0 NULL         NULL      NULL
NULL                            0
...
```

The second SELECT in the UNION doesn't add new data (1 is generally not equal to 0) but prevents the identity property from being propagated.

You can also do it with a SELECT *:

```
1> select * into mycopy from mytable_with_id UNION select * from
mytable_with_id where 1=0
2> go
1> sp_help mycopy
2> go
...
 Column_name Type Length Prec Scale Nulls Default_name Rule_name
Access_Rule_name Computed_Column_object Identity
 ----------- ---- ------ ---- ----- ----- ------------ --------- ------------
---- ---------------------- ----------
 id          int       4 NULL  NULL     0 NULL         NULL      NULL
NULL                            0
```

```
 something    char        1 NULL  NULL      0 NULL          NULL       NULL
NULL                             0
```

Reset the identity counter on a table

When working with an identity column, a counter per table is maintained. This counter is used to generate new values for the identity column of the table.
When you load many data into a table as a test and then truncate the table, the identity counter is not reset. So if you inserted 1000000 rows, truncated the table and inserted an entry, you'd then the value 1000001 for the new entry.

If you want to reset the identity counter for a table in order to have the next value of the identity column be 1 instead, you can change the identity_burn_max attribute for the table e.g.:

```
sp_chgattribute my_table, 'identity_burn_max', 0, '0'
```

Please note that this command creates an exclusive table lock on the table. The lock is not kept for long but this means that the command will be blocked by any read or write lock on the table.

The last parameter is the identity counter you want to set. So if you have kept the entries with the identity values 1 to 10 and deleted the rest, you'd have to set the identiy_burn_max attribute to 10:

```
sp_chgattribute my_table, 'identity_burn_max', 0, '10'
```

If you try to set it to a lower value (i.e. a value lower than the maximum value already in use in the table), sp_chgattribute will fail, refusing to update the attribute because you then risk having duplicate values in there.

You can work around it by directly setting the attribute using dbcc:

```
dbcc set_identity_burn_max(my_database, my_table, 'my_new_value')
```

Also note that if what you want is to actually create an identity gap, all you have to do to increase the counter is to allow identity inserts on the table, insert a higher value and delete it:

```
SET IDENTITY_INSERT my_table ON
go
INSERT INTO my_table(my_identity_column,...) VALUES (1785,...)
go
SET IDENTITY_INSERT my_table OFF
go
DELETE FROM my_table WHERE my_identity_column=1785
go
```

Query optimization and statistics

Analyze the join order selected by the query optimizer

If the query optimizer doesn't choose the query plan you'd expect, and basically would expect tables to be joined in a different order than you'd have expected, you need to get more details about what the optimizer has considered and why it's chosen a given query plan.

First to see what the optimizer chooses for a given query, you should execute the following before executing the query:

```
set showplan on
```

If your query is causing a huge load, you should consider executing it with the noexec option. This will cause the optimizer to show you the query plan, it would use but not execute it:

```
set showplan on
go
set noexec on
go
```

After executing the query, you can set back both options:

```
set noexec off
go
set showplan off
go
```

The order is important since you cannot set showplan back to off while noexec is on.

This will also show you, whether you are performing any table scans and what the expected total cost for the query is.

Now you know which tables are processed in which order and how the joins are performed. Very often, you get a different order on two systems depending on the number of rows, distribution of data, how exact the statistics are... It might also use different indexes on different systems.

If what you're after is why a given index has been chosen for a table, you should execute the following (before setting the noexec option):

```
dbcc traceon(3604, 302)
```

3604 makes sure that the trace output is sent to the client preventing it from filling up your error log.
302 prints information regarding which indexes were considered and why a given index was chosen.

You should check in the output of dbcc trace 302 that all where clauses in the query have been evaluated. If it is not the case the optimizer considered it as not being a valid search argument. This will give you a first possible cause of the problem.

Now that you know which indexes were chosen and why, you will often also want to know why the tables were joined in a particular order. For this you should use the following:

```
dbcc traceon(3604, 310)
```

It will show you the number of tables in join and the number of tables considered at a time. It then prints the first plan considered by the optimizer, and then each plan which is considered cheaper (so iteratively showing you the currently selected plan during the analysis). It writes the heading "NEW PLAN" for each of these plans (it's important if you combine 310 with 317 which displays the rejected plans with another heading). It also prints information about the number of rows expected as input and output of a join, the costs involved (as well expected logical and physical IO).

If the plan you'd have expected to be chosen is in the list, you're done here. You see why the optimizer tought the other plan is better and can start looking for a way to convince the optimizer that the other plan would have been better. Often, the optimizer is not just wrong but has wrong input data (statistics...).

If the plan you'd have expeceted is not there, it's one of the plans which was rejected because the optimizer thought it already had a better plan. So you need to display also rejected/ignored plans:

```
dbcc traceon(3604, 310, 317)
```

The 317 trace basically displays the same info but for not selected plans. It uses the heading "WORK PLAN" instead of "NEW PLAN". This produces a very large output (can easily be 20 MB of text).

If you cannot find any better way to have the optimizer think the way you want it to think, you can use forceplan:

There is a way to force the optimizer to choose a certain plan. Specifically, the join order specified by the query itself, in the "from" clause.

```
set forceplan on
```

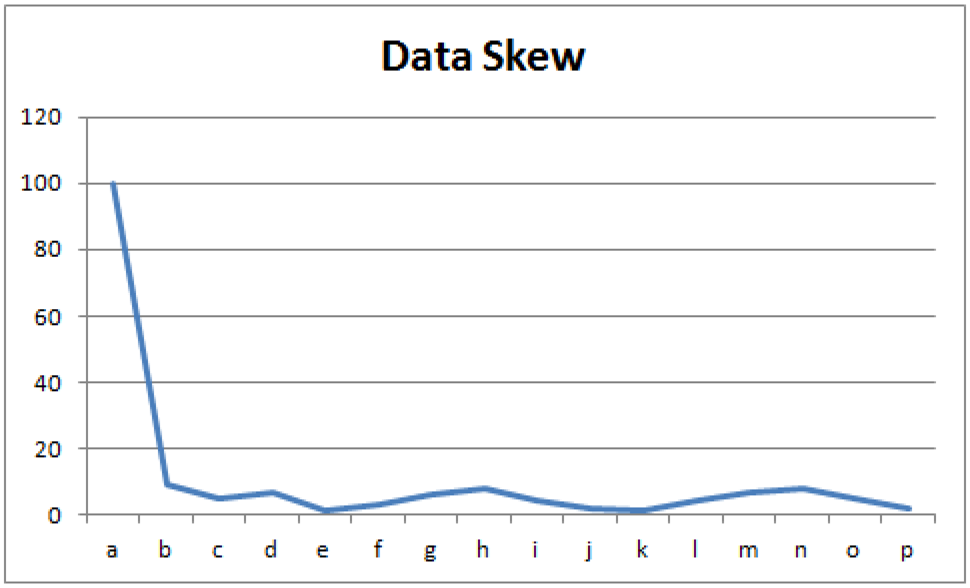
This forces the optimizer to use the exact join order defined in the list of tables in the FROM clause of the query. But this is only useful if you have an influence on the order of table in the FROM clause. I usually do not like using forceplan but sometimes it's the only solution you have (especially if you don't have endless time to solve a performance issue).

Please note that it only tells Sybase which join order to use, not which join types.

Data skew and query plans

In databases, data skew means that many values occupy a small number of rows each and a few values occupy many rows each. So basically it is an asymmetry in the distribution of the data values.

The distribution of data would look like this:



Here a short example to illustrate it:

We have an employee table containing all our employees world-wide (and since we're in this example a quick-ass multinational corporation, we have millions of employees). Now assume we also want to track in this table employees of external companies we are contracting. Additionally we want to also track for which company they are actually working. So we add a table called "company" and reference the ID of this company in our employee table.

The table structure would look like this:

```
create table employee(
id varchar(32),
name varchar(32),
company_id varchar(32))
go
create unique index employee_id_index on employee(id)
go
create index employee_company_index on employee(company_id)
go
create table company(
id varchar(32),
name varchar(32))
go
create unique index company_id_index on company(id)
go
create index company_name_index on company(name)
go
```

Now let's assume we want to get the list of all employees working for company "ext2":

```
select e.name from employee e, company c where e.company_id=c.id and
c.name="ext2"
```

This statement would result in the following query plan:

```
    STEP 1
        The type of query is SELECT.

  FROM TABLE
  company
  c
        Nested iteration.
  Table Scan.
  Forward Scan.
  Positioning at start of table.
  Using I/O Size 4 Kbytes for data pages.
  With LRU Buffer Replacement Strategy for data pages.

  FROM TABLE
  employee
  e
        Nested iteration.
  Table Scan.
  Forward Scan.
  Positioning at start of table.
  Using I/O Size 32 Kbytes for data pages.
  With LRU Buffer Replacement Strategy for data pages.
```

So basically ASE decides to perform a table scan on both tables. The density information for the company_id column leads the optimizer into thinking that it will probably have to through all data pages anyway when using the index on company_id and that it's then actually cheaper to skip the index and directly read the data pages.

The solution to this issue is to modify the statistics in order to change the total density of a column to be equal to the range density:

```
sp_modifystats employee, company_id, REMOVE_SKEW_FROM_DENSITY
```

After that the statement above results in this query plan:

```
    STEP 1
        The type of query is SELECT.

  FROM TABLE
  company
  c
        Nested iteration.
  Table Scan.
  Forward Scan.
  Positioning at start of table.
  Using I/O Size 4 Kbytes for data pages.
  With LRU Buffer Replacement Strategy for data pages.

  FROM TABLE
  employee
  e
        Nested iteration.
  Index : employee_company_index
  Forward Scan.
  Positioning by key.
  Keys are:
    company_id ASC
  Using I/O Size 4 Kbytes for index leaf pages.
  With LRU Buffer Replacement Strategy for index leaf pages.
  Using I/O Size 4 Kbytes for data pages.
  With LRU Buffer Replacement Strategy for data pages.
```

Which is much faster. Of course if the company name we search for is "benohead corp." (our company) instead of "ext2", this query plan would actually be slower than going directly for a table scan. So the optimizer is actually kind of right. You just need to know how what kind of query you perform and if you see that you only check employees for the external companies then it makes sense to modify the density information.

You will have to repeat this operation (modifying the statistics) after every update statistics since updating the statistics will undo the REMOVE_SKEW_FROM_DENSITY.

Note that this problem doesn't occur anymore (or less often ?) with ASE 15. So it's mostly a problem when using ASE 12.5.4 or lower or ASE 15 with the compatibility mode switched ON (e.g. because you have a legacy application which performs really bad with the new optimizer...). With the ASE 15 optimizer, you get the following query plan even though without modifying the density information:

```
    STEP 1
        The type of query is SELECT.



        |ROOT:EMIT Operator (VA = 3)
        |
        |   |NESTED LOOP JOIN Operator (VA = 2) (Join Type: Inner Join)
        |   |
        |   |   |SCAN Operator (VA = 0)
        |   |   |  FROM TABLE
        |   |   |  company
        |   |   |  c
        |   |   |  Table Scan.
        |   |   |  Forward Scan.
        |   |   |  Positioning at start of table.
        |   |   |  Using I/O Size 4 Kbytes for data pages.
        |   |   |  With LRU Buffer Replacement Strategy for data pages.
        |   |
        |   |   |SCAN Operator (VA = 1)
        |   |   |  FROM TABLE
        |   |   |  employee
        |   |   |  e
        |   |   |  Index : employee_company_index
        |   |   |  Forward Scan.
        |   |   |  Positioning by key.
        |   |   |  Keys are:
        |   |   |    company_id ASC
        |   |   |  Using I/O Size 32 Kbytes for index leaf pages.
        |   |   |  With LRU Buffer Replacement Strategy for index leaf pages.
        |   |   |  Using I/O Size 32 Kbytes for data pages.
        |   |   |  With LRU Buffer Replacement Strategy for data pages.
```

But the ASE 15 optimizer also has other problems of its own...

Updating statistics

In ASE, statistics are used by the Optimiser to determine the appropriate Query Plan. Wrong or out-of-date statistics can cause the wrong query plan to be selected which can lead to massive performance issues. It is thus very important to consider statistics in regular database maintenance activities and make sure they are updated on a regular basis.

There are a few ways in which you can make sure that the statistics are up-to-date and support the query optimizer in choosing the optimal query plan:

- the update statistics command
- the update index statistics command
- the update all statistics command
- the update table statistics command
- the sp_modifystats stored procedure

In the end of the article, I'll also shortly discuss the performance considerations required when choosing the appropriate command.

Update Statistics

Using the update statistics command, you can update the statistics about the distribution of key values for a table, an index on the table, specific columns of the table or on partitions.

Only reference the table

If you only reference a table name:

```
update statistics mytable
```

it will update statistics for the leading columns of all indexes on the table. The leading column of a composite index is the first column in the index definition. This is the most important one since the index cannot be used if only the second column is used in the WHERE clause but not the first one.

Here's an example of such an index:

```
create index mytable_comp_ind on maytable(col1, col2)
```

In this case, col1 is the leading column for this index.

Explicitely reference an index

If you explicitely reference an index on the table:

```
update statistics mytable mytable_comp_ind
```

it will update the statistics for the leading column of the specified index i.e. it will update the statistics on col1.

Explicitely reference a partition

If you explicitely reference a partition on the table:

```
update statistics mytable partition mypartition
```

it will update the density information on all columns of all indexes of the partition. It will additionally also create histograms for all leading columns of indexes of the partition. You can also provide a list of column names. In this case it will create histograms for the first column and densities for the composite columns.

Note that only local indexes of the partition are considered. Global indexes are not considered. Also note that updating the statistics for a partition also updates the global statistics i.e. only local indexes are considered but global statistics are updated with the gathered data.

Explicitely reference columns

If you explicitely reference a list of columns:

```
update statistics mytable (col1, col2)
```

it will create histograms for the first referenced column and density information for all column groups with the first column as leading column.

Note that performing such an update statistics has two consequences:

1. A table scan is performed
2. A sort is performed, meaning that you need to have enough space in tempdb

So this update statistics does have a high impact on the performance of the server and requires more space in tempdb.

Also note that in many cases statistics are mostly useful for columns referenced by indexes (especially as leading columns). Updating statistics for other columns create an overhead. But in some cases it is required and better than using update all statistics.

Update Index Statistics

The command update index statistics update the statistics of all columns of an index:

```
update index statistics mytable mytable_comp_ind
```

The difference between this command and:

```
update index statistics mytable mytable_comp_ind
```

is that the latter focuses on the lead column of the index. The former creates histograms not only for the leading column of the index but for all columns specified in the index definition.

You can also update the statistics for all columns of all indexes on the table by omitting the index name:

```
update index statistics mytable
```

Note that it is not exactly the same as calling update index statistics for each index on the table since the latter updates the statistics of columns referenced by multiple indexes multiple times. So if you intend to update the statistics for all indexes, it's more performant to omit the index name than issuing one command per index.

Here also you can specify a data or index partition.

From a performance point view, this means that:

1. An index scan will be performed to update the statistics of the leading column of the index
2. For the other columns, a table scan for each of them will be required followed by a sorting based on a table in tempdb

This is basically the same problem as when using update statistics on columns which are not the leading column of an index.

Update All Statistics

This command creates statistics for all columns of the specified table:

```
update all statistics mytable
```

A partition can also additionally be specified.

From a performance point of view, this means that:

1. An index scan will be performed for each column which is the leading column of an index
2. A table scan will be performed for each other table, creating a work table in tempdb which will then be sorted

Update Table Statistics

The update table statistics is kind of a different beast. It doesn't update the data in systabstats but only table or partition level statistics in sysstatistics i.e. it does not affect column level statistics.

Here the syntax:

```
update table statistics mytable
```

Or with a partition name:

```
update table statistics mytable mypartition
```

Modifying Statistics

sp_modifystats is a system stored procedure which can be used to update density information for a table of a column group.

There are two uses of this command.

Modify density information

You can use the MODIFY_DENSITY parameter to change the cell density information for columns:

```
sp_modifystats mytable, "col1%", range, absolute, "0.5"
```

This will set the range cell density to 0.5 for the column col1 as well as for all column group with density information having col1 as leading column.

The column group parameter can either contain:

- "all" meaning that all column groups should be updated
- A column name meaning that only the density information for this column should be updated
- A list of columns meaning that the density information of this specified column group should be updated
- A column name or column list including wildcards as shown above

Note that the next update statistics will reset the density information. So it's a good idea to always call sp_modifystats after an update statistics.

Handling skew in data

Please refer to my previous post regarding data skew, their effect on queries and how to handle them: Data skew and query plans.

Performance

Updates of statistics are a required maintenance activities if you want to keep your queries fast. On the other hand, updating statistics also has a non-negligeable impact on performance and load.

In general the leading column of indexes are the critical ones. So you need to make sure they are always up-to-date.
But in some cases it does make sense to have up-to-date statistics also for other columns: If you have a WHERE clause also containing this column not being part of an index and 99% of the values in this column are the same, this will greatly impact how joins are done. Without statistics on this column, a default distribution will be assumed and the wrong join might be selected.

So I'd recommend:

1. Using "update statistics" instead of the other commands to optimize the leading columns of indexes
2. Using "update statistics" on specific columns which are not leading column in an index, in case you see that ASE chooses the wrong join because it assumes a default distribution of data and it is not the case. Use it only if really required as it creates a huge load on the system
3. Avoid using "update all statistics". It generally makes more sense to use dedicated commands to update what needs to be updated

More information regarding sampling, histogram steps and the degree of parallelism will be added later...

Force loading in cache

You can load an entire index into Sybase's cache by executing the following:

```
select count(*) from <tablename> (index <indexname> lru)
```

index <indexname> forces the optimizer to use the specified index.
lru tells Sybase to use the LRU Buffer Replacement Strategy i.e. keep the data in cache after the execution of the statement.

This will perform an index scan and load the index data in cache.

If you want to perform a table scan instead i.e. load the data pages in cache, you can write the table name instead of the index name. But if you have an unclustered index with the same name as the table, it will be loaded in cache instead to make sure that the table is loaded, you can issue the following statement:

```
select count(*) from <tablename> (index 0 lru)
```

Please note that this will not load the text and image columns of the table.

In order to check whether the whole table is in cache, you can do the following:

```
select DBName+".."+ObjectName, CachedKB*100/TotalSizeKB as percentage from
monCachedObject where IndexID = 0 order by percentage desc
```

This lists all tables in cache and the percentage of the data pages loaded.

Indexes

Find all tables in a database with a unique index

The following statement will return a list of all user tables in the current database which have a unique index:

```
SELECT o.name FROM sysobjects o, sysindexes i WHERE o.id = i.id
AND o.type='U' AND i.status & 2 = 2 GROUP BY o.name ORDER BY
o.name
```

If you're also interested in the index name, use the following:

```
SELECT o.name, i.name FROM sysobjects o, sysindexes i WHERE o.id
= i.id AND o.type='U' AND i.status & 2 = 2 ORDER BY o.name
```

Why does changing the locking scheme take so long ?

If you've already changed the locking scheme of a huge table from allpages locking (which was the locking scheme available in older versions of ASE) to datapages or datarows locking, you've noticed that it does take a long time and keeps you CPU pretty busy.

The reason is that when switching between allpages locking and data locking basically means copying the whole table and recreating the indexes.

Here are all the steps involved in such a change:

- All rows of the table are copied into new data pages and formatted according to the new locking scheme.
- All indexes are dropped and the re-created.
- The old table is removed.
- The information regarding this table in the system tables are updated.
- The table counter is updated to force recompilation of query plans.

Note that this is also the reason why some people use this switching back and forth as a way to defragment a table.

The first two steps are the one taking the most time. It's difficult to estimate the time required for this. But you can get an idea by checking the size of the data and indexes for this table. This can be done using sp_spaceused:

```
1> sp_spaceused report
2> go
 name   rowtotal reserved   data      index_size unused
 ------ -------- ---------- --------- ---------- --------
 report 1837164  8377528 KB 687468 KB 7676644 KB 13416 KB

(1 row affected)
(return status = 0)
```

It doesn't tell you how much time is needed but if you do it on different table, you could assume the time needed is almost proportional to the size of data+indexes.

Note that switching between datapages and datarows locking schemes only updates system tables and is thus pretty fast.

Get info regarding indexes and which segment they are located on

The following SQL statement displays all indexes on user tables in the current database, with the related table, whether it's a clustered index and on which segment it is located:

```
select so.name as TableName, si.name as IndexName, ss.name as SegmentName,
case when (si.status & 16 = 16 OR si.status2 & 512 = 512) then 'clustered'
else 'non-clustered' end as 'Clustered' from sysindexes si, syssegments ss,
sysobjects so where si.segment=ss.segment and si.id=so.id and so.type='U' and
si.name != so.name order by so.name, si.name, ss.name
```

Please note that clustered indexes are usually located on the default segment.

Combine with the output of sp_estspace, you can find out what the required size of the index segment should be.

Space used by the indexes on a table

In order to find the space used by a table (data and indexes) you can execute the following:

```
1> sp_spaceused mytable
2> go
 name    rowtotal reserved  data    index_size unused
 ------- -------- --------- ------- ---------- ------
 mytable 20612    103320 KB 7624 KB 95216 KB   480 KB

(1 row affected)
(return status = 0)
```

If you see that the indexes take much more space than expected, you can also use 1 as a second parameter to the sp_spaceused stored procedure:

```
1> sp_spaceused mytable,1
2> go
 index_name             size reserved unused
 ------------------ -------- -------- ------
 my_index_1           612 KB   736 KB 124 KB
 my_index_2          2840 KB  2864 KB  24 KB
 my_index_3           504 KB   536 KB  32 KB
 my_index_4          3016 KB  3032 KB  16 KB
 my_index_5          1144 KB  1152 KB   8 KB
 tmytable           87064 KB 87288 KB 224 KB
 mytable_key_index     36 KB  7712 KB  52 KB

(1 row affected)
 name    rowtotal reserved  data    index_size unused
 ------- -------- --------- ------- ---------- ------
 mytable 20612    103320 KB 7624 KB 95216 KB   480 KB
(return status = 0)
```

The tmytable entry are the text and image fields.

Performance issue with clustered index on allpages locked tables

When a table has a clustered index, ASE makes sure that all rows are physically stored in the order defined by the columns on which you have the clustered index. There can only be one clustered index on a given table as ASE cannot store the data with two different orders.

You can either explicitely created using the following command:

```
create clustered index myclusteredindex on mytable(mycolumn)
```

Or by defining a primary constraint:

```
create table mytable(
        mycol1 int not null,
        mycol2 int not null,
        primary key (mycol1)
)
```

Not that if you defined a primary key, the index name is automatically generated and you cannot drop the index using the DROP INDEX command but need to use the DROP CONSTRAINT command.

If the table has a data lock scheme, the table will be reorganized when the clustered index is created but the order of rows will not be further updated. If the table has an allpages lock scheme, then ASE will make sure that the order is maintained.

Note that although clustered indexes (especially when defined through primary key constraints) are often created as unique indexes, it doesn't have to be the case.

Reading from a table with allpages lock and a clustered index using the keys of the clustered index as criteria is almost always faster than without the clustered index. But writing to the table is slower since ASE needs to maintain the order. This can create huge performance issues when working with huge tables with many updates on the index keys or many inserts/deletes. In some cases (I observed a case on a table with 28 million entries), committing or rolling back changes on such a table can cause many physical IOs. If this is done in a transaction also involving updates on other tables this could cause many other connections to blocked. In the case I had observed, it took up to 30 minutes to finish some rollbacks. My assumption was that it is because ASE needs to reorganize the whole index which involves reading an writing many pages. In this case dropping the primary key constraints solved the problem. You can just replace the clustered index by a non-clustered one.

So I'd recommend in general not to use clustered index on huge allpages tables except if you are 100% sure that you need the extra read performance. The penalty while writing can be so huge that it cannot be compensated by the extra read speed.

Locks when creating indexes

Got the following question from a colleague:

We have a table with allpages lock scheme. A nonclustered index is added to this table (which has millions of entries). During the creation of the index, many blocked delete commands are observed. Would having another lock scheme (data page lock or data row lock) help ?

Answer:

The locking scheme of the table doesn't matter in this case. When an index is created, a table lock is always created. If the index is clustered, then the lock is exclusive i.e. SELECT, INSERT, UPDATE and DELETE operations are blocked. Otherwise it is only a shared lock i.e. INSERT, UPDATE and DELETE statements are still blocked but SELECT statement are not blocked. In this case it is a nonclustered index but for a DELETE statement it makes no difference, the statement is still blocked.

Devices and segments

Find the devices containing fragments of a database

If you need to find out which devices contain fragments of a database and their physical location, you can use the following SELECT statement:

```
select sda.name, sde.name, sde.phyname from master..sysdatabases sda,
master..sysdevices sde, master..sysusages su where su.dbid=sda.dbid and
su.vdevno=sde.vdevno and sda.name = @dbname
```

where @dbname is the name of you database.

This returns 3 columns:

- The name of the database
- The name of the device
- The physical location of the device

I use this when dropping proxy databases I'm creating for some temporary checks. I want to drop the database and drop their related devices and delete the files.
I have called all my proxy databases tempdb_old1, tempdb_old2,... So I'd get the information I require for my cleanup activity like this:

```
1> select sda.name, sde.name, sde.phyname from master..sysdatabases sda,
master..sysdevices sde, master..sysusages su where su.dbid=sda.dbid and
su.vdevno=sde.vdevno and sda.name like 'tempdb_old%'
2> go
 name                           name                           phyname
 ------------------------------ ------------------------------ --------------
-----------------------------------------------------------------------------
------------------------------------
 tempdb_old1                    tempdb_proxy_dev1
/db/proxy_tempdb1.dat
 tempdb_old2                    tempdb_proxy_dev2
/db/proxy_tempdb2.dat
 tempdb_old3                    tempdb_proxy_dev3
/db/proxy_tempdb3.dat
 tempdb_old4                    tempdb_proxy_dev4
/db/proxy_tempdb4.dat

(4 rows affected)
```

dsync and directio

dsync

First, dsync has no effect on raw devices (i.e., a device on a raw partition) and on devices on Windows operating system (i.e., it only affects Unix/Linux operating systems).

ASE opens a database device file of a device with the dsync setting on, using the operating system dsync flag.With this flag, when ASE writes to the device file, both the written data must be physically stored on disk before the system call returns.

This allows for a better recoverability of the written data in case of crash: If the writes are buffered by the OS and the system crashes, these writes are lost. Of course, this only handles OS level buffering. The data could still be in the disk write cache and get lost...

The drawback of dsync is that it costs performance (because the writes, even if buffered by the OS, are guaranteed to go to the disk before the operation finishes).

It should be noted that dsync doesn't mean that there is not asynchronous I/O. It just means that when you write synchronously or check for whether the asynchronous I/O was performed, you'll only get the response that the write is completed once the data are effectively on the physical disk.

Please also note that a cache and a buffer are different.

dsync is always on for the master device: the performance of writes there is not critical and it's important that it can be fully recovered.

On the other hand, it is common to turn off dsync on devices of databases which do not need to be recovered like the tempdb.

directio

directio is basically a way to get a way to perform I/O on file system devices in a similar way to raw devices i.e. the OS buffer caches are bypassed and data are written directly to disk.

But directio does not guarantee that the writes will only return after all data have been stored on disk (just that data will not go into caches). But it since the OS buffer caches is bypassed, it does provide a pretty good recoverability.

directio provides better write performance than sync (especially if the device is stored on a SAN). On the other hand, dsync is faster on devices for read operations. So transaction log devices are very good candidates for directio (or for raw devices).
Also on newer Linux kernels dsync provides awful performance and you should then rather use directio than dsync.

Note that directio is not supported on HP-UX.

For the tempdb devices you should use neither dsync nor directio (as you do not need the recoverability at all).

Size of data and log segments for all databases

Use the following SQL statement in order to get information about the size and usage of the data and log segments in all databases:

```
select db_name(d.dbid) as db_name,
ceiling(sum(case when u.segmap != 4 then u.size/1048576.*@@maxpagesize end ))
as data_size,
ceiling(sum(case when u.segmap != 4 then size - curunreservedpgs(u.dbid,
u.lstart, u.unreservedpgs) end)/1048576.*@@maxpagesize) as data_used,
ceiling(100 * (1 - 1.0 * sum(case when u.segmap != 4 then
curunreservedpgs(u.dbid, u.lstart, u.unreservedpgs) end) / sum(case when
u.segmap != 4 then u.size end))) as data_used_pct,
ceiling(sum(case when u.segmap = 4 then u.size/1048576.*@@maxpagesize end))
as log_size,
ceiling(sum(case when u.segmap = 4 then u.size/1048576.*@@maxpagesize end) -
lct_admin("logsegment_freepages",d.dbid)/1048576.*@@maxpagesize) as log_used,
ceiling(100 * (1 - 1.0 * lct_admin("logsegment_freepages",d.dbid) / sum(case
when u.segmap in (4, 7) then u.size end))) as log_used_pct
from master..sysdatabases d, master..sysusages u
where u.dbid = d.dbid  and d.status != 256
group by d.dbid
order by db_name(d.dbid)
```

This will display 7 columns:

- The name of the database
- The total size of the data segments of this database (in MB)
- The usage of the data segments of this database (in MB)
- The usage percentage
- The size of the log segments of this database (in MB). This columns will be empty if there is no separated log segment i.e. this database has mixed log/data segment
- The usage of the log segments of this database (in MB). This columns will be empty if there is no separated log segment i.e. this database has mixed log/data segment
- The usage percentage. This column will contain no value if there is neither a separate log segment nor a mixed log segment (e.g. for the dbcc database). If there is a mixed data/log segment, it will display the same percentage as the data segment usage.

A few details about the query:

@@maxpagesize returns the server's logical page size. It's basically the same value you'd get by using the following:

```
select low from master.dbo.spt_values where number = 1 and type = "E"
```

So multiplying a number of pages by @@maxpagesize returns the corresponding number of bytes.

1048576 is the number of bytes in a megabyte. Dividing by 1048576 thus converts all number of bytes in number of megabyte. The dot after 1048576 is used so that we implicitly convert those integers to floats. Here an example of why we need this dot:

- 1048575 / 1048576 * 4096 returns 0 as 1048575 / 1048576 is 0 when doing some pure integer arithmetic
- 1048575 / 1048576. * 4096 returns 4095.99606784

When computing the values you have to make sure that you avoid arithmetic overflow which would happen e.g. if you multiplied by @@maxpagesize before dividing by 1048576.

The values of master..sysusages.segmap mean the following:

- 3: Data stored on this segment
- 4: Log stored on this segment
- 7: Since 7=4+3, both log and data stored on this segment

That's why we exclude segmap = 4 when computing values for the "data" columns, consider only segmap = 4 for the columns related to the size of the log segment or it's usage. But we do consider both segmap=4 and segment=7 for the usage percentage of the log since when both are on the same segment, a full segment would also indicate a full log.

curunreservedpgs returns the number of free pages in the specified piece of disk. The third parameter (we provide here sysusages.unreservedpgs) is returned instead of the value in memory when the database is not opened i.e. not in use.

lct_admin with "logsegment_freepages" as the first parameter returns the number of free pages in a dedicated log segment

Proxy databases and tables

Create a proxy for a remote database

Note that all strings in bold+italic should be changed to something which is relevant for your setup.

First you have to make the remote server known locally:

```
sp_addserver "logical_name" , ASEnterprise,
"xxx.xxx.xxx.xxx:2055"
```

You can choose anything as a logical name. This is how the remote server will be called in master..sysservers and the name used by the other

commands we'll use.
If you use another port than 2055, you of course have to change it in the statement above.
"ASEntreprise" is the remote server type and means it is also an ASE server.

You can check whether the remote server has been added properly:

```
SELECT * FROM master.dbo.sysservers WHERE srvname =
"logical_name"
```

Now you need to create an entry in the sysremotelogins table for the remote server. This is done with the stored procedure sp_addremotelogin.

If you have the same users on both server, you can just execute the following to map remote names to local names:

```
sp_addremotelogin logical_name
```

If you want to map all logins from the remote server to a local user name:

```
sp_addremotelogin logical_name, local_user_name
```

If you only want to map a single remote login from a remote user on the remote server to a local use:

```
sp_addremotelogin logical_name, local_user_name,
remote_user_name
```

Now we can create the proxy database:

```
create database proxy_db_name on default = 200M with
default_location = 'logical_name.remote_db_name..' for
proxy_update
```

If you do not want to create it on default but on a new device, you'll need to first create the device.

```
disk init name="proxy_dev",
physname="/var/sybase/ASE/proxy_dev.dat", size="200M"
```

Of course the parent directory of the file which path is set in physname should exist and the appropriate rights should be set.
(you can also added the parameter directio=true to this command if required)

And then create the database:

```
create database proxy_db_name on proxy_dev = 200M with
default_location = 'logical_name.remote_db_name..' for
proxy_update
```

Now you can use the proxy database to access data of all tables in the remote database:

```
select name from proxy_db_name..sysobjects
```

The data are still residing on the remote server so you do not need to do anything when data change. But if the structure changes (i.e. if you add or remove tables or update the structure of a table), you need to update the proxy:

```
alter database proxy_db_name for proxy_update
```

Change the IP address of a remote server

In order to access a remote database, you need to add a remote server and create a proxy database as shown here.

If your remote server gets a new IP address, you can of course drop the proxy database and remote server and recreate them. But I didn't want to do that, so just update the sysservers table in the master database:

```
1> sp_configure 'allow updates to system tables', 1
2> go
 Parameter Name                 Default              Memory Used Config Value
Run Value            Unit                 Type
 ------------------------------ -------------------- ----------- ------------
-------- -------------------- -------------------- --------------------
 allow updates to system tables          0                     0            1
1         switch               dynamic

(1 row affected)
Configuration option changed. ASE need not be rebooted since the option is
dynamic.
Changing the value of 'allow updates to system tables' does not increase the
amount of memory Adaptive Server uses.
(return status = 0)
1> update master..sysservers set srvnetname='192.168.230.236:2055' where
srvid=4
2> go
(1 row affected)
1> sp_configure 'allow updates to system tables', 0
2> go
 Parameter Name                 Default              Memory Used Config Value
Run Value            Unit                 Type
```

```
 ------------------------------ -------------------- ----------- ------------
-------- -------------------- -------------------- --------------------
 allow updates to system tables          0                    0           0
0         switch               dynamic

(1 row affected)
Configuration option changed. ASE need not be rebooted since the option is
dynamic.
Changing the value of 'allow updates to system tables' does not increase the
amount of memory Adaptive Server uses.
(return status = 0)
```

In order to update a system table like sysservers you first need to allow such changes. After modifying sysservers, you can disable such updates again as shown above.

Now testing whether I can still refresh my proxy:

```
1> alter database myproxydb for proxy_update
2> go
Msg 11216, Level 16, State 13:
Server 'SYBASE', Line 1:
Internal Ct-Lib/Cs-Lib error 84083972: 'ct_connect(): network packet layer:
internal net library error: Net-Lib protocol driver call to connect two
endpoints failed'.
Msg 11206, Level 16, State 1:
Server 'SYBASE', Line 1:
Unable to connect to server 'SOURCE'.
```

So I thought it was some kind of network connectivity issue, but:

```
# telnet 192.168.230.236 2055
Trying 192.168.230.236...
Connected to 192.168.230.236.
Escape character is '^]'.
```

So I can connect to the remote ASE with telnet, so must have missed something else. In the end, I never figured out why it didn't but did find an alternative way to do this:

```
1> sp_addserver SOURCE, ASEnterprise, "192.168.230.236:2055"
2> go
Msg 17290, Level 16, State 1:
Server 'SYBASE', Procedure 'sp_addserver', Line 328:
There is already a server named 'SOURCE', physical name
'192.168.230.236:2055'.
```

Ok, it doesn't work because I've already updated sysservers manually, so setting it back to the way it was before the update:

```
1> sp_configure 'allow updates to system tables', 1
2> go
 Parameter Name                 Default              Memory Used Config Value
Run Value           Unit                 Type
```

```
 ------------------------------ -------------------- ----------- ------------
-------- -------------------- -------------------- --------------------
 allow updates to system tables          0                    0           1
1         switch               dynamic

(1 row affected)
Configuration option changed. ASE need not be rebooted since the option is
dynamic.
Changing the value of 'allow updates to system tables' does not increase the
amount of memory Adaptive Server uses.
(return status = 0)
1> update master..sysservers set srvnetname='192.168.230.225:2055' where
srvid=4
2> go
(1 row affected)
1> sp_configure 'allow updates to system tables', 0
2> go
 Parameter Name                 Default              Memory Used Config Value
Run Value            Unit                 Type
 ------------------------------ -------------------- ----------- ------------
-------- -------------------- -------------------- --------------------
 allow updates to system tables          0                    0           0
0         switch               dynamic

(1 row affected)
Configuration option changed. ASE need not be rebooted since the option is
dynamic.
Changing the value of 'allow updates to system tables' does not increase the
amount of memory Adaptive Server uses.
(return status = 0)
```

And changing the IP address again using sp_addserver:

```
1> sp_addserver SOURCE, ASEnterprise, "192.168.230.236:2055"
2> go
Changing physical name of server 'SOURCE' from '192.168.230.225:2055' to
'192.168.230.236:2055'
```

That's right! sp_addserver doesn't only add a server but can also change the physical name of a server. Now trying to refresh the proxy again:

```
1> alter database pdir_old for proxy_update
2> go
```

No error message this time ! That was easy (well, once you figure out that sp_addserver is actually sp_add_or_update_server...).

Triggers and stored procedures

Get the text of a trigger or stored procedure

In order to get the text of a trigger, you can use (just like for stored procedure) the sp_helptext stored procedure:

```
sp_helptext mytriggername
```

But there are a few problems with this:

- It's not easy to further use the output of the procedure in further processing. You need to work with the loopback adapter and create a proxy table...
- The text is stored in chunks of 255 characters so if you just execute the procedure and redirect to a file, you get unwanted newlines in there

The text of triggers and stored procedure is stored in the syscomments table:

```
1> sp_help syscomments
2> go
 Column_name Type      Length Prec Scale Nulls Default_name Rule_name
Access_Rule_name Computed_Column_object Identity
 ----------- --------- ------ ---- ----- ----- ------------ --------- --------
-------- ---------------------- ----------
 id          int            4 NULL  NULL     0 NULL         NULL      NULL
NULL                              0
 number      smallint       2 NULL  NULL     0 NULL         NULL      NULL
NULL                              0
 colid       smallint       2 NULL  NULL     0 NULL         NULL      NULL
NULL                              0
 texttype    smallint       2 NULL  NULL     0 NULL         NULL      NULL
NULL                              0
 language    smallint       2 NULL  NULL     0 NULL         NULL      NULL
NULL                              0
 text        varchar      255 NULL  NULL     1 NULL         NULL      NULL
NULL                              0
 colid2      smallint       2 NULL  NULL     0 NULL         NULL      NULL
NULL                              0
 status      smallint       2 NULL  NULL     1 NULL         NULL      NULL
NULL                              0
 partitionid int            4 NULL  NULL     1 NULL         NULL      NULL
NULL                              0
```

The id column is the object id of the trigger or stored procedure. This can be used to make a join with the sysobjects table to have access to the object name:

```
select c.text
from syscomments c, sysobjects o
where o.id=c.id and o.name='mytriggername' order by c.colid
```

With the statement above you still get chunks of 255 characters. Now you need to iterate through the results and store them in a variable:

```
declare trigger_text_cursor cursor
for select c.text
from syscomments c, sysobjects o
where o.id=c.id and o.name='mytriggername' order by c.colid
for read only
go

declare @triggertext varchar(16384)
declare @text varchar(255)

open trigger_text_cursor
fetch trigger_text_cursor into @text

while @@sqlstatus = 0
begin
        set @triggertext=coalesce(@triggertext, '')+@text
        fetch trigger_text_cursor into @text
end

close trigger_text_cursor
deallocate cursor trigger_text_cursor
select @triggertext
go
```

You can then use @triggertext to perform any further processing you need.

coalesce is used so that if a value is null, it will use the empty string. You could also do the same thing using isnull.

Note that it is not possible to declare a TEXT variable. Instead you have to declare a large VARCHAR variable (as done above). The only drawback is that the maximal length of such a variable is 16384 characters. If you have triggers or stored procedures with a longer text, you'll have to implement the loop in a script or program (instead of using a cursor in Transact-SQL).

Retrieve the text of a long trigger or stored procedure

To workaround the limitation above: It could only retrieve up to 16384 characters which is the limit for a varchar variable. Defining a text variable is not possible. Now I have a few triggers longer than 16384 and their text is being truncated.

So what are the options ?

1. Storing the text in a variable is not an option.
2. Creating a dummy table with one text row and appending the text is not option either since you'd then get an error saying you cannot add to a text column (you can only read it and store it, but not append to it).
3. Just selecting all corresponding rows in syscomments is not an option either, since it would split lines

I ended up have a table and storing one row per line in the trigger text. Identifying the lines basically means appending the text to a varchar(16384) variable, searching for a line feed, storing the text before the line feed in the table, doing this recursively until you got all lines currently in the buffer and resuming with the rest.

Here a stored procedure doing this:

```
create procedure print_trigger(@trigger_name varchar(255))
as
begin
        create table tempdb..triggertext (triggertext text)

        declare trigger_text_cursor cursor
        for select c.text
        from pdir..syscomments c, pdir..sysobjects o
        where o.id=c.id and o.name=@trigger_name order by c.colid
        for read only

        declare @triggertext varchar(16384)
        declare @triggertext2 varchar(16384)
        declare @text varchar(255)
        declare @index int

        open trigger_text_cursor
        fetch trigger_text_cursor into @text

        while @@sqlstatus = 0
        begin
                -- Appends each line of the trigger text in syscomments to the
text already gathered.
                set @triggertext=coalesce(@triggertext, '')+@text
                -- Loop through the lines (delimited by a line feed
                select @index=charindex(Char(10),@triggertext)
                while (@index >0)
                begin
                        select @triggertext2=substring(@triggertext,1,@index-1)
                        -- Add each line to the table
                        if (@triggertext2 is not null)
                                insert into tempdb..triggertext (triggertext)
values(@triggertext2)
                        -- Continue with the rest of the string
                        select
@triggertext=substring(@triggertext,@index+1,16384)
                        select @index=charindex(Char(10),@triggertext)
                end
                fetch trigger_text_cursor into @text
        end

        close trigger_text_cursor
        deallocate cursor trigger_text_cursor

        if (@triggertext is not null)
                insert into tempdb..triggertext (triggertext)
values(@triggertext)

        select triggertext from tempdb..triggertext
        drop table tempdb..triggertext
end
```

```
go
```

If you want to execute this with isql and write the output to a file, you additionally do the following:

- call isql with the -b option which disables the display of the table headers output.
- set nocount on to suppress X rows affected messages.
- set proc_return_status off to suppress the display of return statuses when stored procedures are called.
- pipe the result to the following sed command to remove all those annoying trailing spaces (basically removing all spaces before an end of line):

  ```
  sed 's/[[:space:]]*$//'
  ```

Find all triggers on a table

In order to find all triggers on a table (select, delete, insert and update triggers), you can use the stored procedure sp_depends:

```
1> sp_depends tablename
2> go
Things inside the current database that reference the object.
 object                         type
 ------------------------------ ----------------
 dbo.sp1                        stored procedure
 dbo.sp2                        stored procedure
 dbo.sp3                        stored procedure
 dbo.my_del_trigger             trigger
 dbo.my_ins_trigger             trigger
 dbo.my_upt_trigger             trigger
 dbo.another_table_trg          trigger
...
```

There are two problems when using this procedure:

- sp_depends relies on the sysdepends table and there were always problems with this table not being updated properly.
- As you can see above it also shows you all triggers referencing this table even if it's a trigger on another table.

A similar solution to this which also has the problem that it displays too many objects:

```
1> select distinct so.name from sysobjects so, syscomments sc where
so.id=sc.id and so.type='TR' and sc.text like '%tablename%'
2> go
```

In order to reliably find only triggers on a given table, you have to check the sysobjects table:

```
1> select so2.name from sysobjects so1, sysobjects so2 where (so2.id =
so1.deltrig or so2.id = so1.instrig or so2.id=so1.updtrig or
so2.id=so1.seltrig) and so1.name='tablename'
2> go
```

This will return a list of the names of all triggers on the table.

Alternatively, you can also get the names of the triggers using the following statement (without the double join and with one column per trigger instead of one row per trigger):

```
1> select object_name(deltrig), object_name(instrig), object_name(updtrig),
object_name(seltrig) from sysobjects where name='tablename'
2> go
```

If you need to also get the text of the triggers, you can use sp_helptext <trigger_name> or use the method described in Get the text of a trigger or stored procedure.

Note: There is a seltrig column in sysobjects but I'm not aware of a way to actually define such a trigger so it's probably not used in ASE.

Drop all triggers on a table

You can use the following stored procedure to drop all triggers on a table:

```
create procedure drop_all_triggers(@tablename varchar(255))
as
begin
        select so2.name into #deletetriggers from sysobjects so1, sysobjects
so2 where (so2.id = so1.deltrig or so2.id = so1.instrig or so2.id=so1.updtrig
or so2.id=so1.seltrig) and so1.name=@tablename

        declare delete_trigger_cursor cursor
        for select name from #deletetriggers

        declare @trigger_name varchar(255)
        declare @selectString varchar(10000)

        open delete_trigger_cursor
        fetch delete_trigger_cursor into @trigger_name

        while @@sqlstatus = 0
        begin
                set @selectString = 'drop trigger '+@trigger_name
                print @selectString
                exec (@selectString)
                fetch delete_trigger_cursor into @trigger_name
        end

        close delete_trigger_cursor
        deallocate cursor delete_trigger_cursor
end
go
```

It basically does the following:

- Fetch the name of all triggers on the table into a temp table (we have to go through a temp table since by dropping a trigger we modifying the sysobject table and it would mess with the cursor.
- Loop through the trigger names
- And drop each of them using an execute immediate. Somehow you cannot directly call drop trigger...

Just call it like this:

```
1> drop_all_triggers mytablename
2> go
```

Enabling and disabling triggers

If you want to disable a trigger, you can of course just drop it. But when you need it again, you'll have to recreate it. If you just want to disable the trigger because you're copying data to the table and this data has already been preprocessed and the triggered is just getting in the way, you can disable it with the alter table command:

```
1> alter table mytable disable trigger
2> go
```

This will disable all triggers on the table. If you only want to disable a single trigger, you can use the following:

```
1> alter table mytable disable trigger mytriggername
2> go
```

To enable all triggers:

```
1> alter table mytable enable trigger
2> go
```

To enable a specific trigger:

```
1> alter table mytable enable trigger mytriggername
2> go
```

If you want to check whether the triggers are enabled or disabled, you'll have to check the sysobjects entry for this table. The column sysstat2 contains the information you're looking for:

```
select
object_name(instrig) as InsertTriggerName,
case sysstat2 & 1048576 when 0 then 'enabled' else 'disabled' end as
InsertTriggerStatus,
object_name(updtrig) as UpdateTriggerName,
case sysstat2 & 4194304  when 0 then 'enabled' else 'disabled' end as
UpdateTriggerStatus,
object_name(deltrig) as DeleteTriggerName,
case sysstat2 & 2097152  when 0 then 'enabled' else 'disabled' end as
DeleteTriggerStatus
from sysobjects where name='mytable'
```

Keep in mind that the three flags used above are not documented and could by changed by SAP in the future.

To check this for all tables:

```
select convert(varchar(30), name) as TableName,
convert(varchar(30), isnull(object_name(instrig), '')) as InsertTriggerName,
case sysstat2 & 1048576 when 0 then 'enabled' else 'disabled' end as
InsertTriggerStatus,
convert(varchar(30), isnull(object_name(updtrig), '')) as UpdateTriggerName,
case sysstat2 & 4194304  when 0 then 'enabled' else 'disabled' end as
UpdateTriggerStatus,
convert(varchar(30), isnull(object_name(deltrig), '')) as DeleteTriggerName,
case sysstat2 & 2097152  when 0 then 'enabled' else 'disabled' end as
DeleteTriggerStatus
from sysobjects where type='U' and (instrig<>0 or updtrig<>0 or deltrig<>0)
order by name
```

If you use identifiers longer than 30 bytes (which is only possible in ASE 15 and above), you can remove the convert varchar(30).

Executing a stored procedure in another database

Sometimes you need a stored procedure to be executed in a given database. The reason could be that you use functions like drop trigger or create trigger which do not allow you to specify a database. But if you are currently in the context of another database and need to stay there (e.g. because you need to update proxy databases and need to be in the master database for this), you cannot switch databases by using:

```
1> use mydbname
2> go
```

This is not allowed in the scope of a stored procedure.

But what you can do is create a second stored procedure in the other database and call it using the database name:

```
create procedure mymasterprocedure
as
begin
    ...
    exec myotherdb..myprocinotherdb
    ...
end
```

```
go
```

Note that this also works for procedure which are in the sybsystemprocs database (and are thus callable from any database) i.e. you can call any of these procedures with a database name before the procedure name and execute this procedure in the context of another database e.g.:

```
1> exec myotherdb..sp_help mytablename
2> go
```

Other topics

Check whether a temporary table exists

You can check for the existence of non-temporary tables (even in in tempdb) like this:

```
IF EXISTS (SELECT 1 FROM <databasemase>..sysobjects where
type='U' and name='<tablename>' )
```

Unfortunately it doesn't work for temporary tables since temporary tables (e.g. #mytemptable) are created for the current connection and another connection might create another temporary table with the same name. So Sybase will create an entry in tempdb..sysobjects with a name containing some other info (like the spid):

```
create table #1234567890 (i1 int)
```

will be mapped to:

```
ASE pre-15: #1234567890__00000306012214068
ASE 15: #12345678900000306001221406 8
```

So you might do something like this:

```
IF EXISTS (SELECT 1 FROM tempdb..sysobjects where type='U' and
name LIKE '<tablename>%'+convert(varchar(30),@@spid)+'%' )
```

Unfortunately, this will not work since you might have a connection with spid 30 and another with spid 306 and these '%' will mess everything up.

Also note that ASE 12.5 only considers the 13 first bytes of the table name. ASE 15 supports 238 bytes.

Please also note that the way these names are generated is not documented and might thus be changed without notice.

So finding out whether another session has created a temporary table with a given name is possible, but not easy and might be broken with any future version or patch.

But finding out whether such a table has been created in the same session is much easier:

```
IF object_id('<tablename>') IS NOT NULL
```

Note that this also works with global temporary tables (##xxxxx).

Update TEXT/IMAGE columns using dbwritetext

Similar to the writetext command, you can use the dbwritetext function of the Open Client DB Library to update a text or image column.

Here is a short sample how to do it:

```
// Defines the entry point for the console application.
//

#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <sybfront.h>
#include <sybdb.h>

LOGINREC     *login;
DBPROCESS    *dbproc;
DBCHAR       mytextcol[512];

int _tmain(int argc, _TCHAR* argv[])
{
        if (dbinit() == FAIL)
                exit(ERREXIT);

        // Create a login  record and fill it with username, password and
application name.
        login = dblogin();
        DBSETLUSER(login, "myusername");
        DBSETLPWD(login, "mypassword");
        DBSETLAPP(login, "myapplicationname");

        // Open a connection to the server using the login record.
        dbproc = dbopen(login, "myservername");

        // You need to first select the text/image field in order to be able
to get the pointer.
        dbcmd(dbproc, "select mytextcol from mydb..mytable where myuid =
'582289'");
        dbsqlexec(dbproc);
        dbresults(dbproc);
        dbbind(dbproc, 1, STRINGBIND, (DBINT) 0, (BYTE *)&mytextcol);
        dbnextrow(dbproc);

        // This is the new value for the field.
        strcpy(mytextcol, "My update text value.");

        // Now write it with dbwritetext
        dbwritetext (dbproc, "mydb..mytable.mytextcol", dbtxptr(dbproc, 1),
DBTXPLEN, dbtxtimestamp(dbproc, 1), TRUE, (DBINT)strlen(mytextcol), (BYTE
*)&mytextcol);

        // The end...
        dbexit();
        return 0;
}
```

myservername is not the hostname of the server but the name in the Sybase configuration (you can see it using dsedit).
TRUE: it means that the operation should be logged. If you set it to FALSE, the operation will be minimally logged which is suboptimal in case of media recovery (but is faster and doesn't fill the transaction log).

You need to additionally set the following in your visual C++ project:
Additional Include Directories: "C:sybaseOCS-15_0include"
Additional Library Directories: "C:sybaseOCS-15_0lib";"C:sybaseOCS-15_0dll"
Additional Dependencies: libsybct.lib libsybdb.lib

The advantage of using dbwritetext are:

- It's fast.
- It can handle large volumes of data (up to 2GB).

The disadvantages are:

- It does not activate insert or update triggers.
- If you have a timestamp column in the table it's value will not be updated when using dbwritetext

If the program crashes on dbbind, check whether your SQL-Select is right (I had a typo in the selected column name and wasted half an hour wondering why it was crashing).

Find in which table a column name is available

If you have a column name but do not know to which table it belongs to, you can use the syscolumns system table in the appropriate database.
syscolumns contains column metadata:

```
1> sp_help syscolumns
2> go
 Column_name Type        Length Prec Scale Nulls Default_name Rule_name
Access_Rule_name Computed_Column_object Identity
 ----------- ----------- ------ ---- ----- ----- ------------ --------- -----
----------- ---------------------- ----------
```

```
 id          int              4 NULL  NULL     0 NULL         NULL      NULL
NULL                            0
 number      smallint         2 NULL  NULL     0 NULL         NULL      NULL
NULL                            0
 colid       smallint         2 NULL  NULL     0 NULL         NULL      NULL
NULL                            0
 status      tinyint          1 NULL  NULL     0 NULL         NULL      NULL
NULL                            0
 type        tinyint          1 NULL  NULL     0 NULL         NULL      NULL
NULL                            0
 length      int              4 NULL  NULL     0 NULL         NULL      NULL
NULL                            0
 offset      smallint         2 NULL  NULL     0 NULL         NULL      NULL
NULL                            0
 usertype    smallint         2 NULL  NULL     0 NULL         NULL      NULL
NULL                            0
 cdefault    int              4 NULL  NULL     0 NULL         NULL      NULL
NULL                            0
 domain      int              4 NULL  NULL     0 NULL         NULL      NULL
NULL                            0
 name        longsysname    255 NULL  NULL     0 NULL         NULL      NULL
NULL                            0
 printfmt    varchar        255 NULL  NULL     1 NULL         NULL      NULL
NULL                            0
 prec        tinyint          1 NULL  NULL     1 NULL         NULL      NULL
NULL                            0
 scale       tinyint          1 NULL  NULL     1 NULL         NULL      NULL
NULL                            0
 remote_type int              4 NULL  NULL     1 NULL         NULL      NULL
NULL                            0
 remote_name varchar        255 NULL  NULL     1 NULL         NULL      NULL
NULL                            0
 xstatus     int              4 NULL  NULL     1 NULL         NULL      NULL
NULL                            0
 xtype       int              4 NULL  NULL     1 NULL         NULL      NULL
NULL                            0
 xdbid       int              4 NULL  NULL     1 NULL         NULL      NULL
NULL                            0
 accessrule  int              4 NULL  NULL     1 NULL         NULL      NULL
NULL                            0
 status2     int              4 NULL  NULL     1 NULL         NULL      NULL
NULL                            0
 status3     smallint         2 NULL  NULL     0 NULL         NULL      NULL
NULL                            0
 computedcol int              4 NULL  NULL     1 NULL         NULL      NULL
NULL                            0
 encrtype    int              4 NULL  NULL     1 NULL         NULL      NULL
NULL                            0
 encrlen     int              4 NULL  NULL     1 NULL         NULL      NULL
NULL                            0
 encrkeyid   int              4 NULL  NULL     1 NULL         NULL      NULL
NULL                            0
 encrkeydb   sysname         30 NULL  NULL     1 NULL         NULL      NULL
NULL                            0
 encrdate    datetime         8 NULL  NULL     1 NULL         NULL      NULL
NULL                            0
```

The id column contains the object id of the containing table. The name column contains the name of the column.
In order to find the name of the containing table, you can use the sysobjects table in the appropriate database:

```
1> sp_help sysobjects
2> go
 Column_name  Type              Length Prec Scale Nulls Default_name
Rule_name Access_Rule_name Computed_Column_object Identity
 ------------ ----------------- ------ ---- ----- ----- ------------ --------
- ---------------- ---------------------- ----------
 name         longsysname          255 NULL  NULL     0 NULL         NULL
NULL             NULL                         0
 id           int                    4 NULL  NULL     0 NULL         NULL
NULL             NULL                         0
 uid          int                    4 NULL  NULL     0 NULL         NULL
NULL             NULL                         0
 type         char                   2 NULL  NULL     0 NULL         NULL
NULL             NULL                         0
 userstat     smallint               2 NULL  NULL     0 NULL         NULL
NULL             NULL                         0
 sysstat      smallint               2 NULL  NULL     0 NULL         NULL
NULL             NULL                         0
 indexdel     smallint               2 NULL  NULL     0 NULL         NULL
NULL             NULL                         0
 schemacnt    smallint               2 NULL  NULL     0 NULL         NULL
NULL             NULL                         0
 sysstat2     int                    4 NULL  NULL     0 NULL         NULL
NULL             NULL                         0
 crdate       datetime               8 NULL  NULL     0 NULL         NULL
NULL             NULL                         0
 expdate      datetime               8 NULL  NULL     0 NULL         NULL
NULL             NULL                         0
 deltrig      int                    4 NULL  NULL     0 NULL         NULL
NULL             NULL                         0
 instrig      int                    4 NULL  NULL     0 NULL         NULL
NULL             NULL                         0
 updtrig      int                    4 NULL  NULL     0 NULL         NULL
NULL             NULL                         0
 seltrig      int                    4 NULL  NULL     0 NULL         NULL
NULL             NULL                         0
 ckfirst      int                    4 NULL  NULL     0 NULL         NULL
NULL             NULL                         0
 cache        smallint               2 NULL  NULL     0 NULL         NULL
NULL             NULL                         0
 audflags     int                    4 NULL  NULL     1 NULL         NULL
NULL             NULL                         0
 objspare     unsigned smallint      2 NULL  NULL     0 NULL         NULL
NULL             NULL                         0
 versionts    binary                12 NULL  NULL     1 NULL         NULL
NULL             NULL                         0
 loginame     varchar               30 NULL  NULL     1 NULL         NULL
NULL             NULL                         0
 identburnmax numeric               17   38     0     1 NULL         NULL
NULL             NULL                         0
```

```
 spacestate   smallint                 2 NULL   NULL     1 NULL          NULL
NULL             NULL                                0
 erlchgts     binary                   8 NULL   NULL     1 NULL          NULL
NULL             NULL                                0
 sysstat3     unsigned smallint        2 NULL   NULL     0 NULL          NULL
NULL             NULL                                0
```

The id column is basically the same value as syscolumns.id. The name column is the name of the table. For user table, the type column contains 'U'.

So basically you can execute the following to get the table name and column name for columns which name is 'status':

```
select o.name as TableName, c.name as ColumnName
from sysobjects o, syscolumns c
where o.id=c.id and o.type='U' and c.name='status'
```

kscsinit: connectivity library error. Operation: cs_ctx_alloc()

After restarting a Sybase ASE server, the dataserver did not come up again and the following error messages was at the end of the error log:

```
kscsinit: connectivity library error. Operation: cs_ctx_alloc()
```

If you google for this error message, you oddly find many things related to locales. So I tried setting different values for LC_ALL before starting the data servers. But this didn't help.

The next idea was to compare the current SYBASE.cfg with the previous one. And there was one difference:
number of user connections was increased from 170 to 300. But this happened 5 months ago... So it shouldn't be the cause of the problem... Since it was running late, we tried setting it back anyway and there it was, our server was up and running again !

I first wondered whether someone had set it manually in the config file instead of setting it in isql using sp_configure. I was sure that Sybase would tell me than it didn't have enough memory if the value was too high. So I typed in the following in isql:

```
sp_configure 'number of user connections', 300
go
```

And to my surprise ASE just informed me that it was using some more memory with this setting but didn't complain... So it looks like the value was too high to be able to start but it was fine to set it later on. So now we just need to remember that if we restart ASE, we will probably need to reduce the number of user connections and then increase it again using isql... Strange world...

Find and delete duplicates records

If you need to delete duplicate records from a table (e.g. in order to copy the data to a table with a unique index or to add a unique index on this table), you can use one of the following methods.

First it's important to know what kind of duplication we're talking about:

- There is a unique key on this table (e.g. an identity column). The duplicates are entries where the other columns of the rows are identical but the unique key is different.
- The duplicates are 100% identical rows.

In both cases, you need to delete all rows but one, with the same values for X columns. The difference is just that in the first case you can reference each column with a key (i.e. some kind of row ID) and in the second one you can't.

In the first case, identifying the row to keep is quite easy: you need to get the minimum (or maximum) value of your key (let's call it row_id from now on) and delete all rows with the same values for the other columns but a higher (or lower) row_id. Here an example keeping the lowest row_id:

```
DELETE FROM tablename t1 WHERE t1.row_id > (SELECT MIN(t2.row_id) FROM
tablename t2 WHERE t1.xxx=t2.xxx AND t1.yyy=t2.yyy)
```

xxx and yyy being the columns you want to use to identify duplicates.

Note that I assume that we're talking about non-null values. If you also want to handle null values you need to replace:

```
t1.xxx=t2.xxx AND t1.yyy=t2.yyy
```

by:

```
(t1.xxx=t2.xxx OR (t1.xxx is null AND t2.xxx is null)) AND
(t1.yyy=t2.yyy OR (t1.yyy is null AND t2.yyy is null))
```

If you do not have something like a row_id, it's getting a little bit more difficult since you cannot directly reference a row. There are basically two methods to clean up the duplicates:

If you have few duplicate entries, you can use ROWCOUNT to make sure that for each set of duplicate rows you delete all of them except one.

First you need to find out which sets of duplicate entries you have and how many entries you have in each set:

```
SELECT xxx, yyy, COUNT(*) FROM tablename GROUP BY xxx, yyy HAVING COUNT(*) >
1
```

This will for example return:
xxx_value_1 yyy_value_1 5

So you know you have 5 rows with these two values for xxx and yyy. So you need to delete 4:

```
SET ROWCOUNT 4
DELETE FROM tablename WHERE xxx='xxx_value_1' AND yyy='yyy_value_1'
```

And you can repeat it for all values returned by the SELECT statement. Of course this is kind of a lot of manual work and you only want to do it for a few combinations. If you have thousands or millions of duplicates, this is definitely not a good solution.

In this case, you have to create a new table containing only distinct entries and then...

either copy it back in the original table after truncating it:

```
SELECT DISTINCT * INTO #mytemptable FROM tablename
TRUNCATE TABLE tablename
INSERT INTO tablename SELECT * FROM #mytemptable
DROP TABLE #mytemptable
```

or rename the original table and select the distinct entries directly as the original table:

```
sp_rename 'tablename', 'tablename_old'
SELECT DISTINCT * INTO tablename FROM tablename_old
DROP TABLE tablename_old
```

The advantage of this second method is that you save one step. The disadvantage is that you don't have any index/constraints/triggers/... on tablename anymore. You can also do it the other way around, first a SELECT INTO then drop the original table and then sp_rename on the new table (but you still miss a few things like the indexes).

Of course another possibility is to add an identity column so that you can use the first method of this post. After removing the duplicates, you can then remove the identity column.

Create and configure the dbccdb

First you need to execute the system stored procedure sp_plan_dbccdb (you can start it without parameters to consider all database or provide a database name i.e. if you only plan to run the dbcc checks on one database):

```
1> sp_plan_dbccdb mydb
2> go

Recommended size for dbccdb database is 2023MB (data = 2021MB, log = 2MB).

...

Recommended values for workspace size, cache size and process count are:
```

```
dbname                         scan ws  text ws  cache  comp mem  process
count
mydb                           1340M    335M     335M   0K        15
```

This gives you the basic sizes you need to use when creating and configuring the DBCC database (dbccdb).

Now let's create the database:

```
1> create database dbccdb on dbcc_data_dev = 2021, log on dbcc_log_dev = 2
2> go
```

You of course need to have these 2 devices (dbcc_data_dev and dbcc_log_dev) already available and with sufficient space for the database.

Then we'll create a named cache for the new database:

```
1> sp_cacheconfig dbccdb_cache, "336M"
2> go
```

The size of the cache should be chosen to allow for:

- a pool for small IOs (2K on a server with 2K pages...): it needs to be at least 512K since it's the minimum size for a cache pool
- a pool for large IOs (for the size of an extent. An extent being 8 pages, it means that on a server with 2K pages you need to create a 16K pool, on a server with 4K pages a 32K pool...): it's the size shown by sp_plan_dbccdb (under cache).

So in our case it was 335,5 MB, so we chose 336 MB.

The 4K pool is automatically taking up all the configure cached, so we need to reassign some of it for large IOs:

```
1> sp_poolconfig dbccdb_cache, "335M", "32K"
2> go
```

And we bind the cache to the dbccdb database:

```
1> sp_bindcache dbccdb_cache, dbccdb
2> go
```

Then you need to add the scan and text segments in the dbccdb for our mydb databases:

```
1> sp_addsegment mydb_scanseg, dbccdb, dbcc_data_dev
2> go
```

and

```
1> sp_addsegment mydb_textseg, dbccdb, dbcc_data_dev
2> go
```

You can then assign these 2 segments to our database using sp_dbcc_createws (this basically creates the appropriate tables in dbccdb):

```
1> sp_dbcc_createws dbccdb, mydb_scanseg, mydb_scan_ws, scan, "1340M"
2> go
```

and

```
1> sp_dbcc_createws dbccdb, mydb_textseg, mydb_text_ws, text, "335M"
2> go
```

Then you need to set the number of worker processes:

```
1> sp_dbcc_updateconfig mydb, "max worker processes", "15"
2> go
```

Then assign the cache:

```
1> sp_dbcc_updateconfig mydb, "dbcc named cache","dbccdb_cache","335M"
2> go
```

And the two workspaces we've created above:

```
1> sp_dbcc_updateconfig mydb, "scan workspace", mydb_scan_ws
2> go
```

and

```
1> sp_dbcc_updateconfig mydb, "text workspace", mydb_text_ws
2> go
```

You can now use your dbcc checks e.g.:

```
1> dbcc checkstorage mydb
2> go
```

Update: You can check the settings using sp_dbcc_evaluatedb (using the database name e.g. mydb as parameter). You might notice that the suggested cache size returned by this stored procedure is actually much smaller than the one returned by sp_plan_dbccdb. Never found out why there is such a difference (in my case sp_plan_dbccdb reported 335M just the same as for the text workspace and sp_dbcc_evaluatedb reported less than 10 MB...).

Flush messages and result sets

If you have a procedure returning multiple result sets or printing messages and running for quite some time, you usually want to get the intermediate result sets and message back on the client as they are produced. ASE has a buffer where they are stored on their way to the client and you thus always have a delay which duration depends on how fast the buffer gets filled.

To prevent this behavior you can switch in the FLUSHMESSAGE option before executing the procedure:

```
SET FLUSHMESSAGE ON
go
```

You will see that the messages issued with the print command are returned immediately but there is still a delay for the result sets. FLUSHMESSAGE only has the result set returned to the client when a print is executed. This means that you have to execute a print after the selects for which you want the result back immediately e.g.:

```
print ''
```

If you are using the select to return messages e.g.:

```
SELECT "Everything is fine until now."
```

you should convert it to a print (or add an empty print afterwards):

```
print "Everything is fine until now."
```

The only problem with print is that it handles strings and if you want to just return a number, you have to convert it to a string to print it.

Remove rows affected, return status and dashes from isql output

When you execute a procedure from a shell using isql, the output will contain the following additional stuff:

- Number of rows affected
- Dashes over the column of the result sets
- A return status

Since I'm returning SQL statements I want to store in a file and execute later on, I do not want to see all of this.

Here's how I've suppressed all of this:

- The number of rows affected can be suppressed by switching on the nocount option
- The dashes can be suppressed by calling isql with the -b argument
- The return status can be suppressed by switching off the proc_return_status option

Here an example:

```
# isql -Umyusername -Pmypassword -b <<EOT
set nocount on
go
set proc_return_status off
go
exec mystoredprocedure
go
EOT
```

ISNULL vs. COALESCE

In Transact-SQL (as well as on T-SQL on Microsoft SQL Server), you can use both ISNULL and COALESCE to use a default in case you have NULL values:

```
1> select ISNULL(NULL, 123)
2> go

 -----------
         123

1> select COALESCE(NULL, 123)
2> go

 -----------
         123
```

So what are the differences between these two functions:

1. COALESCE is ANSI standard. So if you think you might have to port the code to another DBMS, it's a safer bet.
2. ISNULL means something totally different in other DBMS e.g. MySQL. There it returns a boolean value meaning whether the expression is NULL or not. So this might confuse colleagues coming from different DBMS.
3. COALESCE is harder to spell... After you've mispelled it 20 times you might consider using ISNULL instead !
4. COALESCE can do more than ISNULL. You can provide a list of X expressions and it will return the first one which is not NULL. You can of course write something like ISNULL(expression1, ISNULL(expression2, expression3)) but it's then much more complex to read.
5. From a performance point of view COALESCE is converted to a case statement which seems to be a little bit slower than ISNULL which is a system function. But I guess the difference in performance is so small that it shouldn't be a criterion to go for ISNULL instead of COALESCE.
6. The datatype, scale and precision of the return expression is the one of the first expression for ISNULL. With COALESCE it's more difficult to figure it out since it is determined by datatype hierarchy. You can get the hierarchy with the following query:

   ```
   select name, hierarchy from systypes order by hierarchy
   ```

If you come from Microsoft SQL Server, you should also note that the following does work in SQL Server (recursively concatenating the strings) but will only return the last row in ASE:

```
declare @triggertext varchar(16384)
select @triggertext=coalesce(@triggertext, '')+text
from syscomments c, sysobjects o
where o.id=c.id and o.name=@trigger_name order by c.colid
```

Find the name of the current database

If you need to switch to a given database, you can use the following statement:

```
1> use master
2> go
```

Replacing master by the database name you want to switch to.

If you want to figure out what's the current database, there is unfortunately no @@db, @@dbid or @@dbname global variable. But you can check it over the current process ID:

```
1> select d.name from master..sysdatabases d, master..sysprocesses p where
d.dbid=p.dbid and p.spid=@@spid
2> go
 name
 ------------------------------
 master

(1 row affected)
```

Update: As Peter correctly noticed, there are built-in functions you can use for this too:

```
1> select db_id()
2> go

 ------
      1

(1 row affected)
1> select db_name()
2> go

 ------------------------------
 master

(1 row affected)
1> select user_id()
2> go

 -----------
           1

(1 row affected)
```

```
1> select user_name()
2> go

 ------------------------------
 dbo

(1 row affected)
1> select suser_id()
2> go

 -----------
           1

(1 row affected)
1> select suser_name()
2> go

 ------------------------------
 sa

(1 row affected)
```

Cannot convert after checking with isnumeric

If you have numeric and not numeric values and what to the isnumeric function to find out which values you can convert, pay attention. Isnumeric is not always reliable. Here an example:

```
1> create table benohead(almostnum varchar(10))
2> go
1> insert into benohead values('e')
2> go
(1 row affected)
1> select isnumeric(almostnum) from benohead
2> go

 -----------
           1

(1 row affected)
1> select convert(int, almostnum) from benohead where isnumeric(almostnum) =
1
2> go
Msg 249, Level 16, State 1:
Server 'SYBASE', Line 1:
Syntax error during explicit conversion of VARCHAR value 'e' to a INT field.
```

So in this case isnumeric returns 1 i.e. true. But you still can't convert it to an int. Converting to a float, money or real will also fail:

```
1> select convert(float, almostnum) from benohead where isnumeric(almostnum)
= 1
2> go
Msg 249, Level 16, State 1:
```

```
Server 'SYBASE', Line 1:
Syntax error during explicit conversion of VARCHAR value 'e' to a FLOAT
field.
1> select convert(money, almostnum) from benohead where isnumeric(almostnum)
= 1
2> go
Msg 249, Level 16, State 1:
Server 'SYBASE', Line 1:
Syntax error during explicit conversion of VARCHAR value 'e' to a MONEY
field.
1> select convert(real, almostnum) from benohead where isnumeric(almostnum) =
1
2> go
Msg 249, Level 16, State 1:
Server 'SYBASE', Line 1:
Syntax error during explicit conversion of VARCHAR value 'e' to a REAL field.
```

But converting to a decimal value will work (well it won't throw an error):

```
1> select convert(decimal, almostnum) from benohead where
isnumeric(almostnum) = 1
2> go

 ---------------------
                     0

(1 row affected)
```

Of course, it depends whether you actually mean 0 when writing e in the column... It will not throw an error but might deliver wrong results...

Show active trace flags

If you need to know which trace flags are active, you can use the dbcc traceflags command. Before using it, you show switch on the trace flag 3604 to display the output to standard output (i.e. the console).

```
1> use master
2> go
1> dbcc traceon(3604)
2> go
DBCC execution completed. If DBCC printed error messages, contact a user with
System Administrator (SA) role.
1> dbcc traceflags
2> go
Active traceflags: 3604, 7717

DBCC execution completed. If DBCC printed error messages, contact a user with
System Administrator (SA) role.
```

If you do not switch on the trace flag 3604, you'll see the following:

```
1> dbcc traceflags
```

```
2> go
DBCC execution completed. If DBCC printed error messages, contact a user with
System Administrator (SA) role.
```

Instead of trace flag 3604 you can also use the flag 3605. The ouput will then be writte to the error log:

```
1> use master
2> go
1> dbcc traceon (3605)
2> go
DBCC execution completed. If DBCC printed error messages, contact a user with
System Administrator (SA) role.
1> dbcc traceflags
2> go
DBCC execution completed. If DBCC printed error messages, contact a user with
System Administrator (SA) role.
```

And in the error log:

```
00:00:00000:00027:2012/10/22 17:19:41.28 server  DBCC TRACEON 3605, SPID 27
Active traceflags: 3604, 3605, 7717
```

You'll notice that it also returns 3604 although it is not active in our session (the ouput wasn't written to the console). Actually trace 3604 is a global trace. Switching it off in another session will also disable it in this session. But switching it on in another session, will not have the output displayed on the console for this session. No clue why...

Please also note that you can switch on the trace flags only for a session using the following:

```
1> set switch on 302
2> go
Switch 302 ('print_plan_index_selection') is turned on.
All supplied switches are successfully turned on.
```

This will switch on the trace flag for this session but it will not be visible with dbcc traceflags (even in this session):

```
1> dbcc traceflags
2> go
Active traceflags: 3604, 7717

DBCC execution completed. If DBCC printed error messages, contact a user with
System Administrator (SA) role.
```

Incorrect syntax near 'go'

If you execute a bunch of SQL statement from a shell script (e.g. to create tables/procedures) and get the following error message:

```
Msg 102, Level 15, State 1:
Server 'SYBASE', Procedure 'xxxxxx', Line xx:
Incorrect syntax near 'go'.
```

But it all works fine when executing it in an SQL client (like SqlDbx or ASEIsql), the problem is probably related to the end of line. If you create the SQL file executed by the script on a Windows machine, it will end the lines with Carriage Return and Line Feed (CR+LF) and on a Linux/Unix system, it should only be a Line Feed (LF). After converting the file to Unix format, it should not see the error anymore.

Under Linux, you can convert the file using:

```
# tr -d '\r' < inputfile > outputfile
```

or:

```
# sed -e 's/\r$//' inputfile > outputfile
```

or:

```
# dos2unix inputfile
```

List all tables with a foreign key to a specific table

If you want to know which tables are all referencing a specific table using a foreign key, you can use one the following system tables:

1. sysconstraints and sysreferences
2. syskeys

The first way will only work if you have constraints backing the foreign keys. If you have just foreign keys without any referential constraints, you will not be able to see anything.

Here's how you can use the syskeys system table to get information about foreign keys:

First the following columns of syskeys are relevant to us:

- id: it's the ID of the table having the foreign/primary/common key.
- type: it tells you whether it's a primary, foreign or common key. In our case we're only interested in foreign keys, so type=2.
- keycnt: since this table contains 8 columns with column IDs, it's useful to know how many of them actually contain something.
- depid: it is the ID of the table we want to find foreign keys for.
- key1 to key8: these are the column IDs in the referencing table.
- depkey1 to depkey8: these are the column IDs in the referenced table.

Let's say we want to find all tables with a foreign key to a table called 'report'. We'd then have the following WHERE clause:

```
select ... from syskeys k
where k.type = 2 and k.depid = object_id('report')
```

Meaning: look for foreign keys (type=2) referecing the report table(depid=).

Now we want to get the name of the referencing table:

```
select object_name(k.id)
```

The names of the referencing columns:

```
select  object_name(k.id),
                col_name(k.depid, depkey1)
                +', '+col_name(k.depid, depkey2)
                +', '+col_name(k.depid, depkey3)
                +', '+col_name(k.depid, depkey4)
                +', '+col_name(k.depid, depkey5)
                +', '+col_name(k.depid, depkey6)
                +', '+col_name(k.depid, depkey7)
                +', '+col_name(k.depid, depkey8)
from    syskeys k
where   k.type = 2
and             k.depid = object_id('report')
```

You'll notice the output looks ugly because:

- object_name returns a huge string with many trailing blanks
- the list of columns looks like key, , , , , , ,

In order to make it look nicer, we'll need to trim the strings for trailing spaces, limit the length of the individual strings to 30 characters and remove the extra commas.

For the extra commas, we can use syskeys.keycnt which will tell us how many depkeyX columns are filled. And we can use the fact that substring('blabla', X, Y) will return NULL if X is less than 1. This means we need a function which returns 1 if the column index is less than keycnt and returns 0 or a negative number otherwise. Luckily such a function does exist: sign. It will return -1 if the argument is negative, 0 if it's 0 and 1 if it's positive. So we can use sign(keycnt - columnIndex + 1). If there are 3 columns:

- sign(3 - 1 +1) = 1
- sign(3 - 2 +1) = 0
- sign(3 - 3 +1) = -1
- sign(3 - 4 +1) = -1
- sign(3 - 5 +1) = -1
- sign(3 - 6 +1) = -1
- sign(3 - 7 +1) = -1

- sign(3 - 8 +1) = -1

So we end up with something like this:

```
select  rtrim(substring(object_name(k.id), 1, 30)),
                rtrim(substring(col_name(k.depid, depkey1),sign(keycnt),30))
                +rtrim(substring(', '+col_name(k.depid, depkey2),sign(keycnt-
1),30))
                +rtrim(substring(', '+col_name(k.depid, depkey3),sign(keycnt-
2),30))
                +rtrim(substring(', '+col_name(k.depid, depkey4),sign(keycnt-
3),30))
                +rtrim(substring(', '+col_name(k.depid, depkey5),sign(keycnt-
4),30))
                +rtrim(substring(', '+col_name(k.depid, depkey6),sign(keycnt-
5),30))
                +rtrim(substring(', '+col_name(k.depid, depkey7),sign(keycnt-
6),30))
                +rtrim(substring(', '+col_name(k.depid, depkey8),sign(keycnt-
7),30))
                from    syskeys k
where   k.type = 2
and             k.depid = object_id('report')
```

And if we also want to have the column names of the referenced table:

```
select  rtrim(substring(object_name(k.id), 1, 30)),
                rtrim(substring(col_name(k.depid, depkey1),sign(keycnt),30))
                +rtrim(substring(', '+col_name(k.depid, depkey2),sign(keycnt-
1),30))
                +rtrim(substring(', '+col_name(k.depid, depkey3),sign(keycnt-
2),30))
                +rtrim(substring(', '+col_name(k.depid, depkey4),sign(keycnt-
3),30))
                +rtrim(substring(', '+col_name(k.depid, depkey5),sign(keycnt-
4),30))
                +rtrim(substring(', '+col_name(k.depid, depkey6),sign(keycnt-
5),30))
                +rtrim(substring(', '+col_name(k.depid, depkey7),sign(keycnt-
6),30))
                +rtrim(substring(', '+col_name(k.depid, depkey8),sign(keycnt-
7),30)),
                rtrim(substring(col_name(k.id, key1),sign(keycnt),30))
                +rtrim(substring(', '+col_name(k.id, key2),sign(keycnt-1),30))
                +rtrim(substring(', '+col_name(k.id, key3),sign(keycnt-2),30))
                +rtrim(substring(', '+col_name(k.id, key4),sign(keycnt-3),30))
                +rtrim(substring(', '+col_name(k.id, key5),sign(keycnt-4),30))
                +rtrim(substring(', '+col_name(k.id, key6),sign(keycnt-5),30))
                +rtrim(substring(', '+col_name(k.id, key7),sign(keycnt-6),30))
                +rtrim(substring(', '+col_name(k.id, key8),sign(keycnt-7),30))
from    syskeys k
where   k.type = 2
and             k.depid = object_id('report')
```

Get first or last day of previous, current or next month

If you want to provide some filter possibilities in your application showing the data stored for the previous, the current or the next month. So you basically need to figure out the first and last day of the corresponding month.
In case you cannot or do not want to do it in the application code itself, you can use simple SQL statements to get this info.

First you need to figure out the current date:

```
declare @today datetime
select @today=getdate()
```

Let's start with the current month:

Today is January, 30th. So the first day of the month is January, 1st. Unfortunately, you cannot directly set the day to 1. But you can extract the day from the date, in our case 30 and 1 is 30 - (30 - 1), so:

```
declare @today datetime
select @today=getdate()
select dateadd(dd,-(day(@today)-1),@today)
```

This will return: Jan 1 2013 1:49PM

So basically we have the right date but for comparison purposes, we'd rather have midnight than 1:49pm. In order to do it, you need to convert it to a date string and back to a datetime:

```
declare @today datetime
select @today=getdate()
select convert(datetime, convert(varchar(10),dateadd(dd,-(day(@today)-
1),@today),101))
```

Now we get: Jan 1 2013 12:00AM

if you're only interested in a string containing the date, just drop the outer convert:

```
declare @today datetime
select @today=getdate()
select convert(varchar(10),dateadd(dd,-(day(@today)-1),@today),101)
```

Use another format than 101 if needed. The complete list of date conversion formats can be found here. For example, for the German date format, use 104 (dd.mm.yyyy).

Now let's get the last day of the current month. This is basically the day before the first day of next month.

So first let's get the first day of next month. This is actually just 1 month later than the first day of the current month:

```
declare @today datetime
select @today=getdate()
select convert(varchar(10),dateadd(mm,1,dateadd(dd,-(day(@today)-
1),@today)),101)
```

This returns: 02/01/2013

Now let's just substract one day and we'll get the last day of the current month:

```
declare @today datetime
select @today=getdate()
select convert(varchar(10),dateadd(dd,-1,dateadd(mm,1,dateadd(dd,-
(day(@today)-1),@today))),101)
```

This returns: 01/31/2013

Since we already have the first day of next month, let's get the last day of next month. This is basically the same again but instead of adding 1 month, you add 2 months:

```
declare @today datetime
select @today=getdate()
select convert(varchar(10),dateadd(mm,2,dateadd(dd,-day(@today),@today)),101)
```

This returns: 02/28/2013

Now let's tackle the previous month. The first day of last month is basically the first day of the current month minus 1 month:

```
declare @today datetime
select @today=getdate()
select convert(varchar(10),dateadd(mm,-1,dateadd(dd,-(day(@today)-
1),@today)),101)
```

This returns: 12/01/2012

And then the last day of previous month. It is one day before the first day of the current month:

```
declare @today datetime
select @today=getdate()
select convert(varchar(10),dateadd(dd,-(day(@today)),@today),101)
```

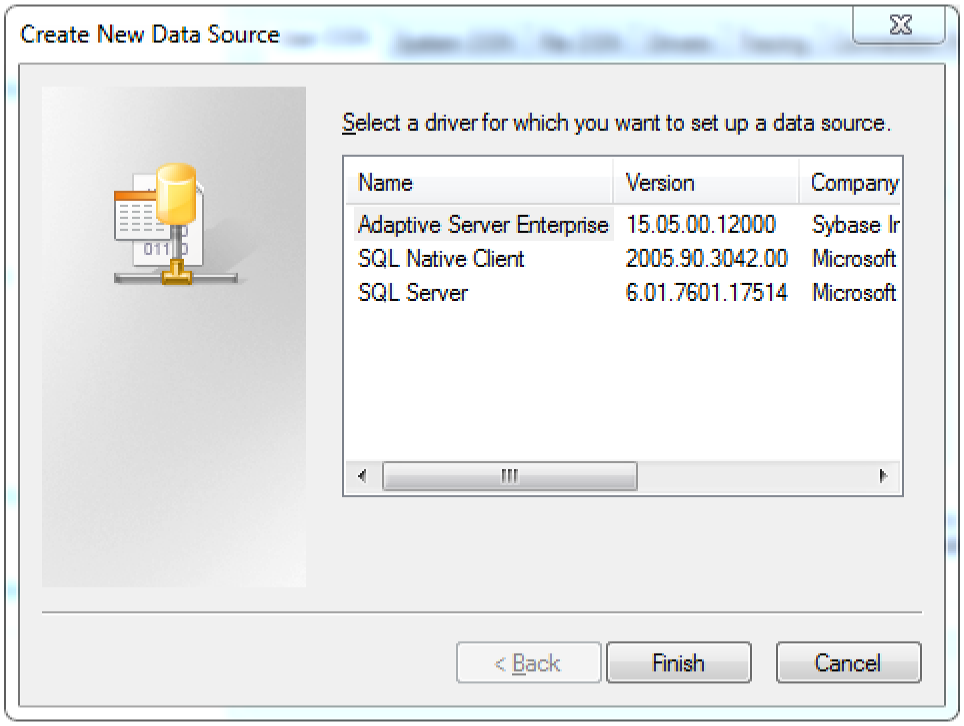
This returns: 12/31/2012

Access the database with C# using ODBC

I needed to write a very short C# program to access a Sybase ASE database and extract some information.

First had to download the appropriate version of ASE. It contains a directory called \archives\odbc. There is a setup.exe. Just run it.

Now there a new driver available:



Create New Data Source

Select a driver for which you want to set up a data source.

Name	Version	Company
Adaptive Server Enterprise	15.05.00.12000	Sybase Ir
SQL Native Client	2005.90.3042.00	Microsoft
SQL Server	6.01.7601.17514	Microsoft

There is no need to add a data source to access ASE from an ODBC connection using C#. Just went there to check whether the driver was properly installed.

Then just create a program connecting to ASE using the following connection string:

```
Driver={Adaptive Server
Enterprise};server=THE_HOSTNAME;port=2055;db=THE_DB_NAME;uid=sa;pwd=THE_SA_PA
SSWORD;
```

If you omit the db=... part, you'll just land in the master database.

With this connection, you can then execute statements.

Here a sample code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.Odbc;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            String errorMsg;
            OdbcConnection con = Connect("sa", "sa.pwd", "2055",
"192.168.190.200", "mydb", out errorMsg);
            Console.WriteLine(errorMsg);
            if (con != null)
            {
                Console.WriteLine("In database {0}", con.Database);
                OdbcCommand command = con.CreateCommand();
                command.CommandText = "SELECT name FROM sysobjects WHERE
type='U' ORDER BY name";
                OdbcDataReader reader = command.ExecuteReader();
                int fCount = reader.FieldCount;
                for (int i = 0; i < fCount; i++)
                {
                    String fName = reader.GetName(i);
                    Console.Write(fName + ":");
                }
                Console.WriteLine();
                while (reader.Read())
                {
                    for (int i = 0; i < fCount; i++)
                    {
                        String col = reader.GetValue(i).ToString();
                        Console.Write(col + ":");
                    }
                    Console.WriteLine();
                }
                reader.Close();
                command.Dispose();
                Close(con);
                Console.WriteLine("Press any key too continue...");
                Console.ReadLine();
            }
        }

        private static OdbcConnection Connect(String strUserName, String
strPassword, String strPort, String strHostName, String dbName, out String
strErrorMsg)
        {
            OdbcConnection con = null;
            strErrorMsg = String.Empty;
            try
            {
                String conString = "Driver={Adaptive Server
Enterprise};server=" + strHostName + ";" + "port=" + strPort + ";db=" +
dbName + ";uid=" + strUserName + ";pwd=" + strPassword + ";";
```

```
                con = new OdbcConnection(conString);
                con.Open();
            }
            catch (Exception exp)
            {
                con = null;
                strErrorMsg = exp.ToString();
            }

            return con;
        }

        private static void Close(OdbcConnection con)
        {
            con.Close();
        }
    }
}
```

That was really easy ! Not that it'd have been more difficult with Java or PHP, but I'd have expected to waste ours making mistakes and trying to debug it...

Find long lasting connections

In order to find which connections have been open for a long time, you can use the following SELECT statement:

```
SELECT
   spid,
   sl.name as 'login',
   sd.name as 'database',
   loggedindatetime,
   hostname,
   program_name,
   ipaddr,
   srl.remoteusername as 'remotelogin',
   ss.srvname as 'remoteservername',
   ss.srvnetname as 'remoteservernetname'
FROM
   master..sysprocesses sp,
   master..syslogins sl,
   master..sysdatabases sd,
   master..sysremotelogins srl,
   master..sysservers ss
WHERE
   sp.suid>0
   AND datediff(day,loggedindatetime,getdate())>=1
   AND sl.suid = sp.suid
   AND sd.dbid = sp.dbid
   AND sl.suid *= srl.suid
   AND srl.remoteserverid *= ss.srvid
ORDER BY
   loggedindatetime
```

Here a few explanations:

- sp.suid>0 filters the ASE processes which run with suid=0
- datediff(day,loggedindatetime,getdate())>=1 filters in only processes which logged in date/time is more than 1 day old
- if the login is a remote login, the remote login and remote server information will be filled. It will contain null otherwise

Convert seconds to a human readable duration

Let's say you get a number of seconds and want to convert it to a string so that it's human readable e.g. saying that something took 15157 seconds is probably not a good idea and it'd make more sense to say it took 4 hours 12 minutes and 37 seconds.

Here's a very simple stored procedure to perform this conversion:

```
create procedure hours_min_sec(@seconds int)
as
begin
select right('00'+convert(varchar(2), floor(@seconds / 3600)),
2)+':'+right('00'+convert(varchar(2),floor(@seconds / 60) % 60),
2)+':'+right('00'+convert(varchar(2),@seconds % 60), 2)
end
go
```

Note that the you may see a line wrap here but actually everything after begin and before end should be on one line.

You can use it like this:

```
1> hours_min_sec 15157
2> go

 --------
 04:12:37

(1 row affected)
(return status = 0)
```