Procurement Desktop – Defense®

Guide to Database Corruption

July 13, 2001

This document contains proprietary and confidential information related to the Procurement Desktop Defense (PD²) product of American Management Systems, Inc. (AMS), as defined in the Software License Agreement (SLA) between AMS and the Department of Defense (DoD), at Section J, Attachment #6, of Contract Number N68939-96-D-0010. This information includes, but is not limited to, icons and software screen prints. Distribution of this document is restricted to employees of the DoD or to third parties who require access on behalf of the DoD and who have executed an appropriate non-disclosure agreement as described in the SLA.

Table of Contents

1. Introduction 3
2. Page and Object Allocation Storage Concepts 4
2.1 The Object Allocation Map (OAM) 5
2.2 Page Linkage 5
3. Table Consistency DBCC Commands 6
3.1 DBCC checkcatalog 6
3.1.1 Syntax and Example 6
3.2 DBCC checktable 7
3.2.1 Syntax and Example 7
3.3 DBCC checkdb 8
3.3.1 Syntax and Example 8
4. Page Allocation DBCC Commands 9
4.1 DBCC checkalloc 9
4.1.1 Syntax and Example 9
4.2 DBCC indexalloc 11
4.2.1 Syntax and Example 11
4.3 DBCC tablealloc 12
4.3.1 Syntax and Example 12
4.4 DBCC textalloc 13
4.4.1 Syntax and Example 13
5. Other DBCC Commands 14
5.1 DBCC traceflags 14
5.1.1 Syntax and Examples of Some Trace Flags 14
5.2 DBCC help 14
5.2.1 Syntax and Example 14
5.3 DBCC Page 15
5.3.1 Syntax and Example 15
5.4 DBCC dbrepair 15
5.4.1 Syntax 15
6. Indexes 16
6.1 Example 1: Drop and Recreate Index (w/ Name) 16
6.2 Example 2: Drop and Recreate Index (w/o Name) 18
6.3 Rebuilding an Index Using Sybase Central 19
7. Rebuilding a Corrupt Table using 'select into' 20
8. Trigers 25
8.1 Verifying if a Table Has Triggers 25
8.1.1 Using SQL Advantage 25
8.1.2 Using Sybase Central 25

AMS Proprietary – Not for Disclosure Outside the Government

1. Introduction

Database Corruption can occur for a variety of reasons. The most common cause of database corruption in the user community comes from hardware or operating system failure, which leads to tables becoming corrupt.

The Database Consistency Checker (DBCC) is a set of utility commands used to check the logical and physical consistency of a database. It checks for errors in index or table page allocations. These commands can detect and sometimes correct errors before they affect PD^2 usage. In the event of hardware or other system failure, DBCC commands can assist in determining the extent of possible damage.

Some DBCC commands (checkcatalog, checkalloc, checkdb and textalloc) should be used as part of regular database maintenance and at minimum should be run on a weekly basis. Ideally, they should be executed on a daily basis before a database is backed up.

When DBCC commands are executed against all tables within a database, it might be difficult to find errors within the log. The majority of times, errors are printed at the very bottom of the output. However, error messages can also be hidden elsewhere within the output. When reviewing the output, the first thing you should do is examine the output at the bottom to see if any errors appear. You should then search the entire output for the following terms: 'Server Message', 'Table Corrupt', 'Drop and re-create', 'Msg', and 'Error'.

Some errors from DBCC's output may indicate severe database consistency problems while others may not be so urgent. Some problems are easily resolved by running dbcc commands that have the fix option or simply rebuilding an index. Others may require a table to be rebuilt or in an extreme case, the user may have to restore from a backup or totally rebuild the database.

For more information regarding the Database Consistency Checker go to the Sybase support web site http://www.sybase.com/support. This site has helpful information for dealing with corruption

Note: This paper has been provided to assist you with understanding and analyzing your DBCC output. If you have any questions or concerns regarding the output after running the DBCC commands please contact the SPS Helpdesk.

AMS Proprietary – Not for Disclosure Outside the Government

2. Page and Object Allocation Storage Concepts

In order to have a good understanding of what the Database Consistency Checker does, it is helpful to have a good understanding of the internal storage structures and how Sybase SQL server stores and manages objects and data within the database.

When the 'disk init' command is used to initialize a new database device, the new space is allocated into 256 2K data pages known as **allocation units**. An **allocation page** is the first page of each allocation unit and it tracks the use of all pages in the allocation unit. Occasionally, DBCC's may report corruption with allocation pages. All allocation pages have an object id of 99.

When a table or index requires space, Sybase SQL server allocates a block of eight 2K pages to the object known as an extent. As a result, a 256-page allocation unit contains 32 extents.

- When you create a table or index, Sybase SQL server allocates an extent for the object.
- When you add rows to an existing table, Sybase SQL server allocates another page if existing pages are full and allocates an additional extent if all pages are full.
- When a table or index is dropped, the extents it occupied are deallocated.
- When rows are deleted from a table, so that it shrinks off a page, Sybase SQL server deallocates the page and if the table shrinks off the extent, Sybase SQL server deallocates the extent.

Every time space is allocated or deallocated on an extent, the event is recorded on the allocation page. This provides a fast method for tracking space allocations in the database.

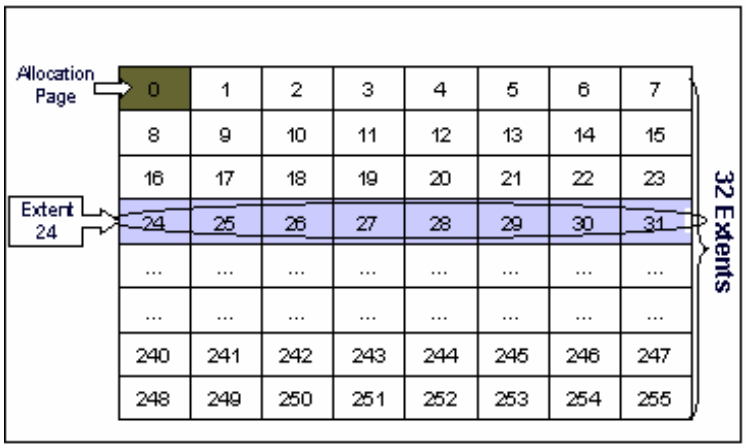


Figure 1:Allocation Unit

AMS Proprietary – Not for Disclosure Outside the Government

2.1 The Object Allocation Map (OAM)

Each table and index has an Object Allocation Map (OAM). The OAM is stored on pages allocated to a table or index and is checked when a new page is needed for the table or index. OAM pages point to the allocation page for each allocation unit where the object uses space. The allocation page tracks information about extent and page usage within the allocation unit.

The OAM is stored on the first page of the first segment (and points to the allocation pages) The allocation pages track the pages used in each extent by all objects with storage space in the allocation unit.

2.2 Page Linkage

Once a page is allocated to a table or index, the page is linked with other pages used for the same object. Each page contains a header that includes the number of the page that precedes it (prev.) and of the number of the page that follows it (next). When a new page is allocated, the header information on the surrounding pages change to point to that page.

AMS Proprietary – Not for Disclosure Outside the Government

3. Table Consistency DBCC Commands

Table Consistency Commands check page linkages and data pointers at both the row and page level.

3.1 DBCC checkcatalog

DBCC checkcatalog checks for consistency problems between system tables and within a system table in a database. It performs the following types of checks for all of the system tables within a database:

- Verifies a table or view in the sysobjects table has at least one row in the syscolumns table
- Verifies that a type in the syscolumns table has a row in the systypes
- Checks that the last **checkpoint** in the syslogs table is valid
- Checks the segment definitions in the syssegments table

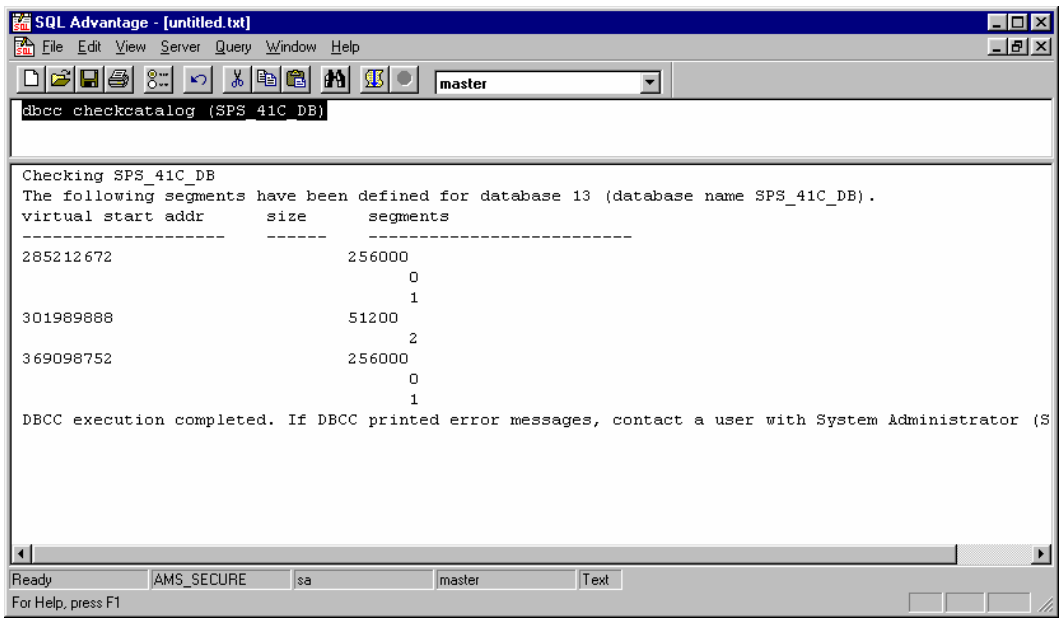
3.1.1 Syntax and Example

The syntax of the dbcc checkcatalog command is:

```
DBCC checkcatalog (<database_name>)
```

<database_name> = name of the database whose consistency is being checked. If the database name is omitted, then the current database will be checked.

Example:



AMS Proprietary – Not for Disclosure Outside the Government

3.2 DBCC checktable

DBCC checktable checks for the following:

- Data pages and indexes are properly linked - making sure previous page, current page, and next page are consistent throughout the page linkage
- Indexes are sorted properly in the correct order
- Consistency of all pointers - making sure the index pointer to the page and row are still valid
- Data rows on each page have entries in an object allocation map (OAM) page and they match their respective locations on the page. Updates the information in the OAM pages that potentially can become inaccurate
- If data rows are not accounted for in the first OAM page for an object, dbcc checktable updates the number of rows on that page

3.2.1 Syntax and Example

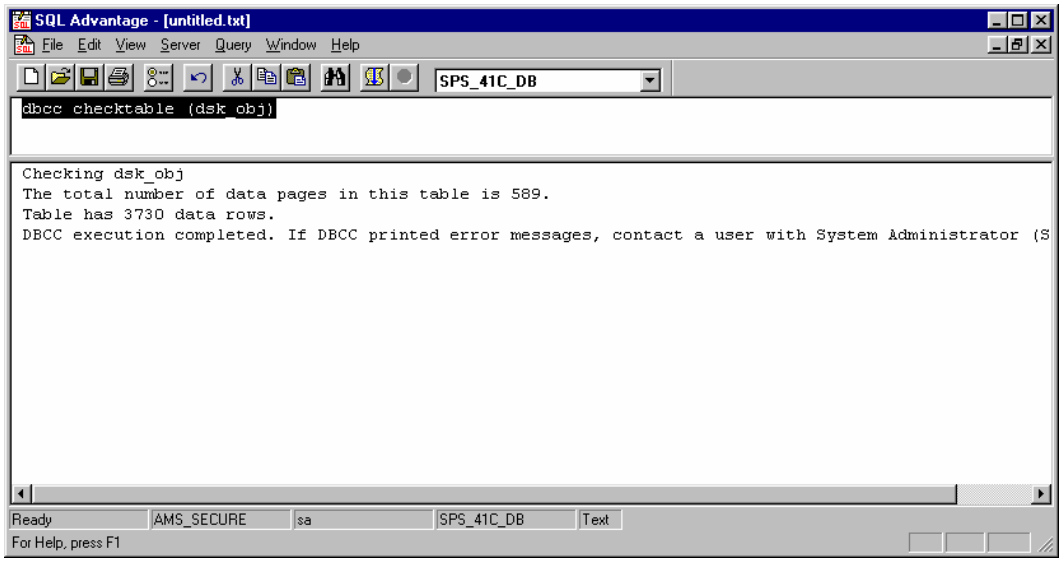
The syntax of DBCC checktable is:

```
DBCC checktable (<table_name> | <table_id> [,skip_ncindex])
```

<table_name> = name of the table being checked, or use

<**table_id**> = id number of the table being checked

The **skip_ncindex** option skips checking the page linkage, pointers, and sort order on nonclustered indexes.



AMS Proprietary – Not for Disclosure Outside the Government

3.3 DBCC checkdb

DBCC checkdb performs a DBCC checktable on each of the tables within a database.

3.3.1 Syntax and Example

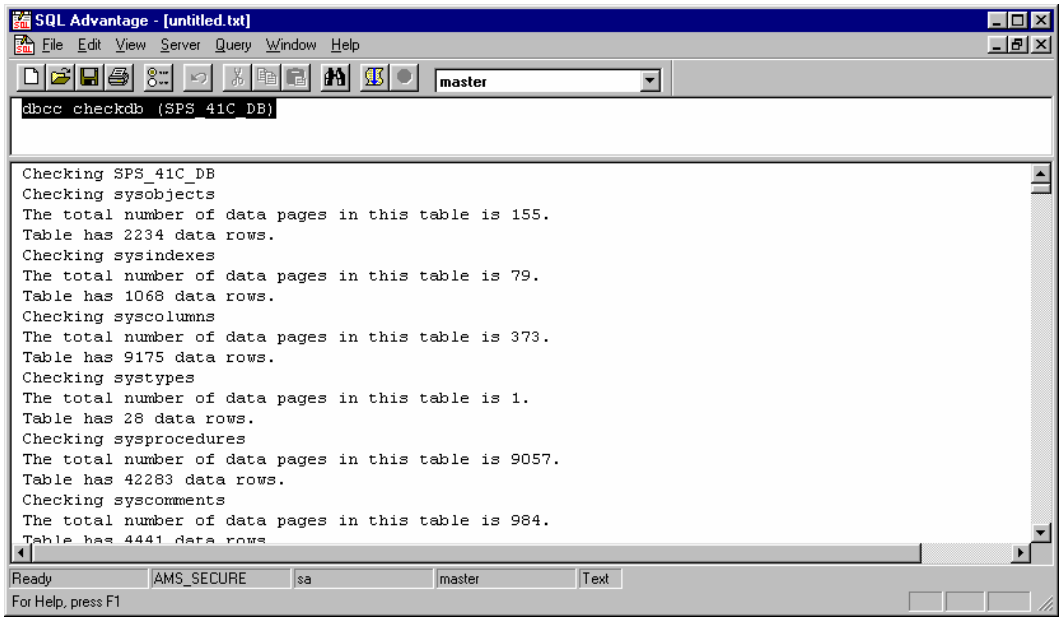
The syntax of DBCC checkdb is:

```
DBCC checkdb (<database_name> [, skip_ncindex])
```

<database_name> = name of the database whose consistency is being checked.

The **skip_ncindex** option skips checking the page linkage, pointers, and sort order on nonclustered indexes.

Example:



Note: Since the DBCC checkdb command accesses a large number of tables, it may take a long time for to complete execution. This command should be executed when it will have the least impact on the production environment.

AMS Proprietary – Not for Disclosure Outside the Government

4. Page Allocation DBCC Commands

Page allocation DBCC commands check allocation structures. It basically checks allocation pages to ensure that pages that are allocated are actually a part of a page linkage, and that pages in a page linkage have been marked as allocated. If an OAM page error does appear, the page allocation dbcc commands with the fix option should be run to attempt to correct them.

4.1 DBCC checkalloc

The DBCC checkalloc command checks the consistency of the page allocation for all tables in a database. It checks to see that:

- All pages have been correctly allocated
- No page is allocated that is not part of a page linkage (i.e. no page is allocated that is not used)
- No page is part of a page linkage that has not been marked as allocated (i.e. no page is used that is not allocated)
- Checks all allocation pages in a database to ensure that they contain valid information
- Checks for an incorrectly set allocation bit for the allocation page on the OAM pages
- Reports the number of pages and extents that have been allocated to the table and any non-clustered indexes
- Reports the amount of space allocated and used. The output consists of data for each table including the number of pages and extents used

4.1.1 Syntax and Example

The syntax for DBCC checkalloc is:

```
DBCC checkalloc (<database_name> [, fix | nofix])
```

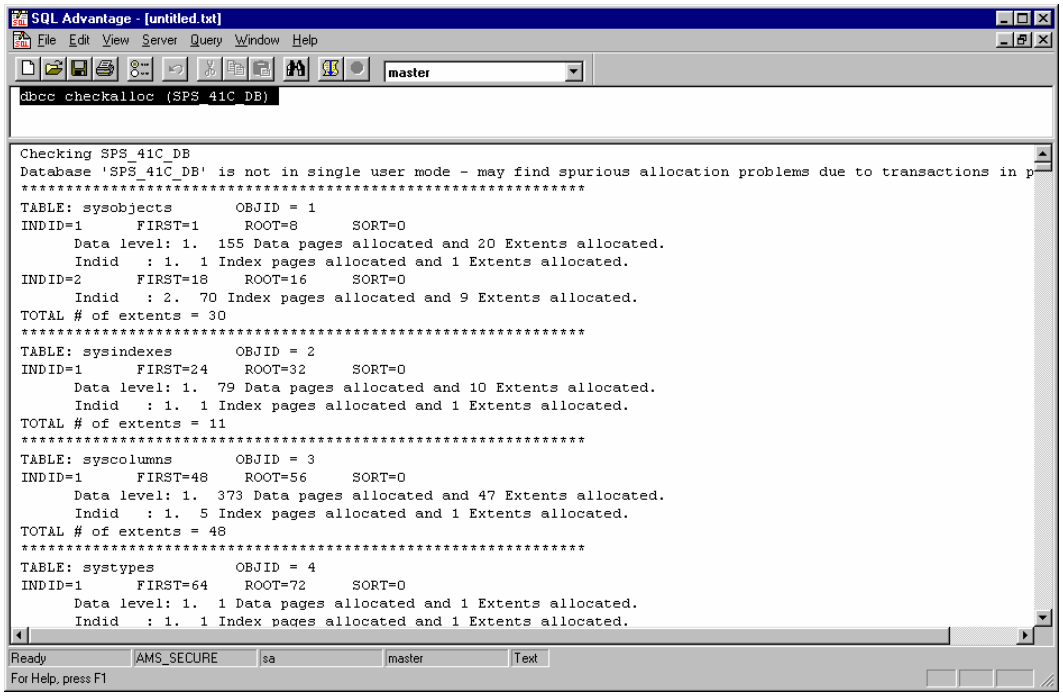
<database_name> = name of the database whose consistency is being checked.

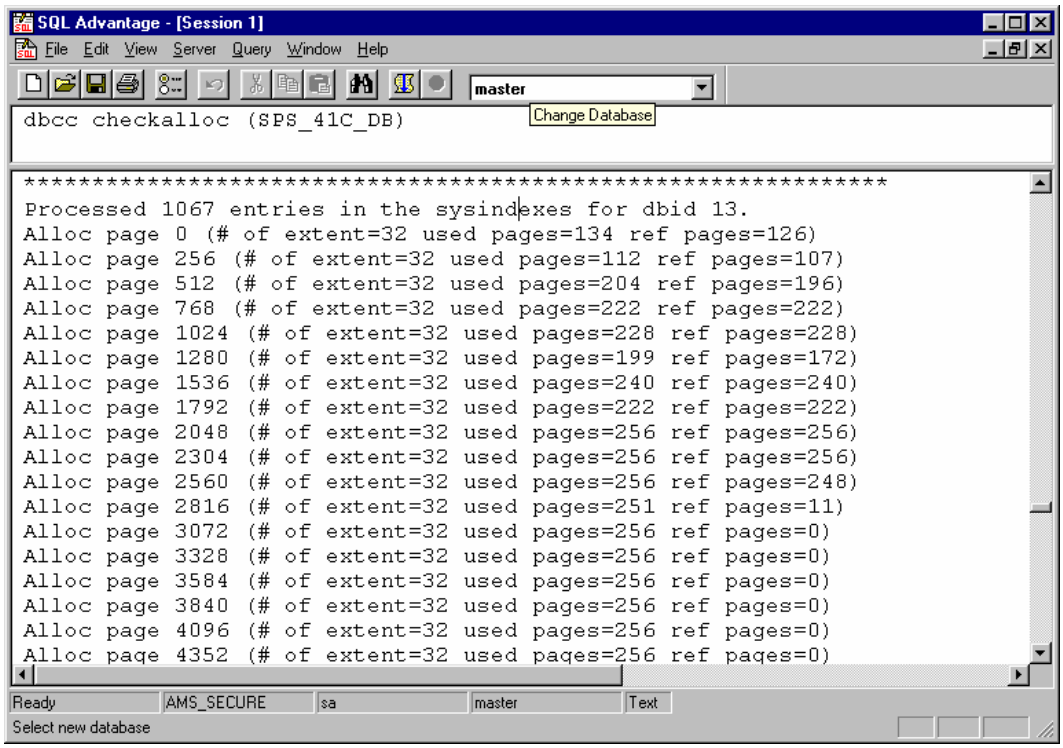
The **fix** option automatically fixes allocation errors. The **nofix** option does not fix allocation errors, but reports the errors that need to be fixed.

Note: The database MUST be in single-user mode if running DBCC checkalloc with the fix option. It might be more useful to use DBCC checkalloc to locate errors and the use DBCC tablealloc to fix errors.

AMS Proprietary – Not for Disclosure Outside the Government

Example of the DBCC checkalloc command:





Note: Since the DBCC checkalloc command accesses a large number of tables, it may take a long time for to complete execution. This command should be executed when it will have the least impact on the production environment.

AMS Proprietary – Not for Disclosure Outside the Government

4.2 DBCC indexalloc

DBCC indexalloc applies to a single index structure. With the full option turned on, it checks that:

- All index pages have been correctly allocated
- No index page that is not part of a page linkage is allocated
- No index page is part of a page linkage that has not been marked as allocated

4.2.1 Syntax and Example

```
dbcc indexalloc (<table_name> | <table_id>, index_id
[, {full | optimized | fast | null}[, fix | nofix]])
```

<table_name> = name of the table being checked, or use

<table_id> = id of the table being checked.

<**index_id**> = id of the index being checked.

Remaining Options:

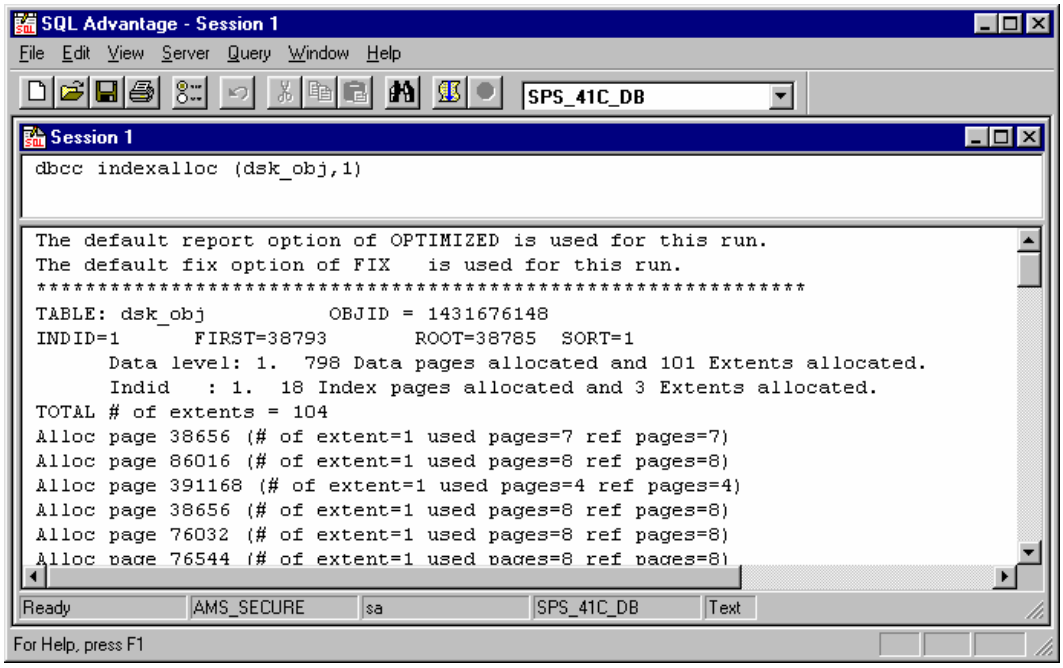
- **full:** Checks all pages for all types of allocation errors. This is the method used by DBCC checkalloc.
- **optimized:** Checks only allocation pages referenced in the OAM pages. This is the default option.
- **fast:** Checks whether all index pages that are part of a page linkage have been allocated.
- **fix:** Fixes allocation errors detected by the consistency check. This is the default option for indexes on user tables.

Note: For indexes on system tables, the database MUST be in single-user mode for the fix option to work.

- **nofix:** Does not fix any allocation errors. This is the default for indexes on system tables

AMS Proprietary – Not for Disclosure Outside the Government

Example:



4.3 DBCC tablealloc

DBCC tablealloc applies to a single table structure and with the full option turned on, it checks that:

- All pages have been correctly allocated
- No page is allocated that is not part of a page linkage
- No page is part of a page linkage that has not been marked as allocated

4.3.1 Syntax and Example

```
dbcc tablealloc (<table_name> | <table_id> [, {full |
optimized | fast | null} [, {fix | nofix ] ])
```

<table_name> = name of the table being checked, or use

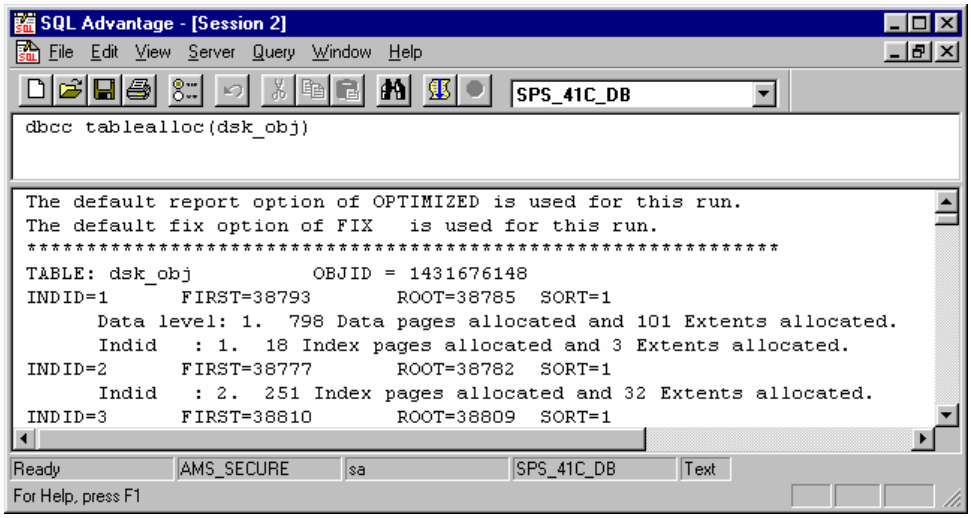
<**table_id**> is the table's id number.

Remaining Options:

- See descriptions under Section 4.2.1

AMS Proprietary – Not for Disclosure Outside the Government

Example:



4.4 DBCC textalloc

DBCC textalloc checks the integrity of a tables text page chain.

4.4.1 Syntax and Example

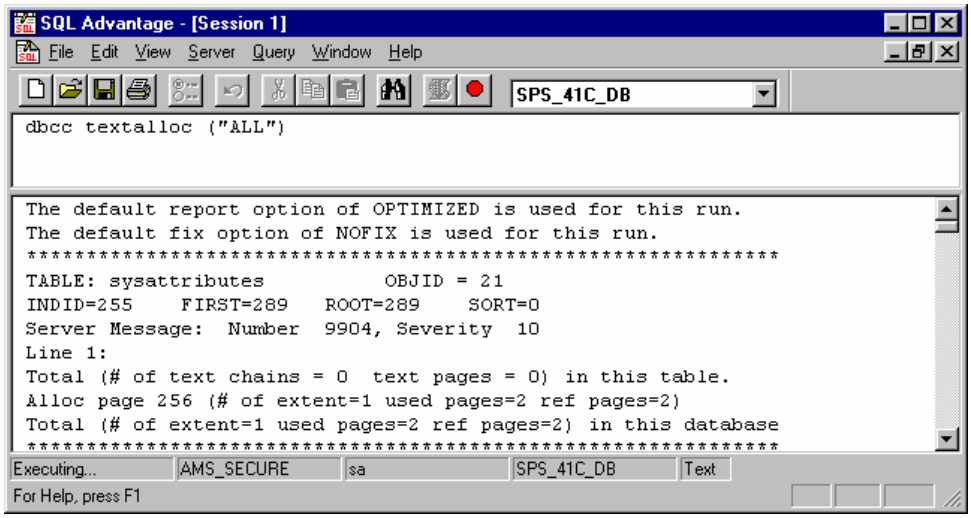
```
DBCC textalloc ("ALL" | table_name [, {full | optimized |
fast | null} [, {fix | nofix ] ] )
```

"ALL" is used to check all tables, or use

<table_name> is the name of the table being checked .

Remaining Options:

- See descriptions under Section 4.2.1



AMS Proprietary – Not for Disclosure Outside the Government

5. Other DBCC Commands

5.1 DBCC traceflags

Traceflags are options that are set during a SQL session. Once set these flags display various information about the SQL command that is being executed.

5.1.1 Syntax and Examples of Some Trace Flags

```
dbcc {traceon | traceoff} (traceflag)
```

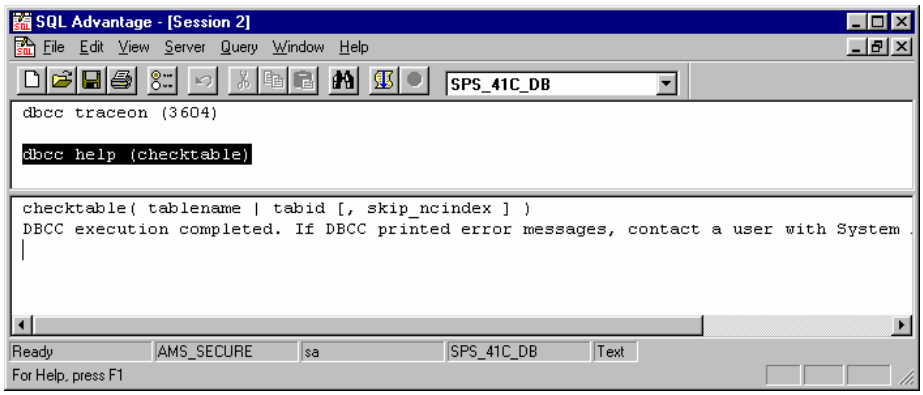
- **302** – Prints information on index selection
- **310** – Prints information on join selection by selecting the first plan chosen and all cheaper plans.
- **317** – Prints complete information on join selection by selecting the first plan and all more expensive plans.
- **1204** – Prints deadlock information. Can also use **sp_configure "print deadlock information", 1**
- **3604** – Directs output from dbcc commands to window where the command is executed
- **3605** – Directs output from dbcc commands to the server error log.

5.2 DBCC help

The DBCC help command displays the syntax for a dbcc command. You need to issue the dbcc traceon (3604) command to send the output from dbcc help to the user terminal.

5.2.1 Syntax and Example

```
dbcc help (dbcc_command)
```



AMS Proprietary – Not for Disclosure Outside the Government

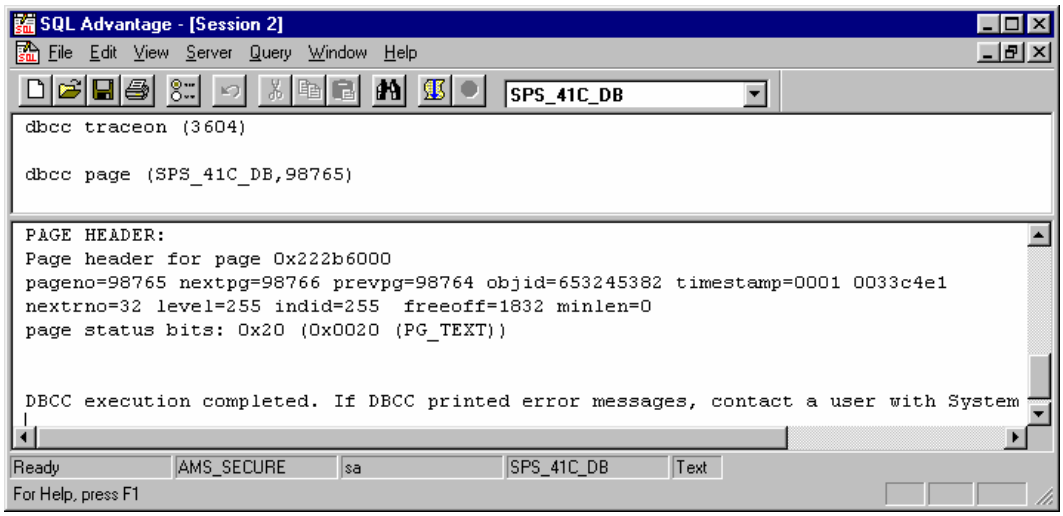
5.3 DBCC Page

The DBCC page command enables you to view a page header and, optionally, the data on a page.

5.3.1 Syntax and Example

```
dbcc page (dbid|dbname, pagenum [, printopt={0|1|2|3} [,
cache={0|1} [, logical={1|0} [, cachename | -1 ]]]])
```

Note: Before running the dbcc page command, you will need to run dbcc traceon (3604) command to send the output to the screen.



5.4 DBCC dbrepair

The DBCC dbrepair command drops a damaged database when the drop database command does not work. The drop database command will not work on a database that cannot be recovered (i.e. database is marked suspect). This command must be issued from the master database and no users can be in the database when it is dropped.

5.4.1 Syntax

```
dbcc dbrepair (database_name, option = {dropdb | fixindex |
fixsysindex}, table, indexid)
```

Note: Please contact the SPS Helpdesk before issuing this command. This command should only be used as a last resort after all attempts to recover the database have failed.

AMS Proprietary – Not for Disclosure Outside the Government

6. Indexes

DBCC output may indicate that an index should be dropped and recreated. Below are some examples of the messages from the dbcc checkdb output that indicate the need to recreate indexes. The procedure for rebuilding indexes is also outlined.

6.1 Example 1: Drop and Recreate Index (w/ Name)

The following is an example of output from a DBCC command:

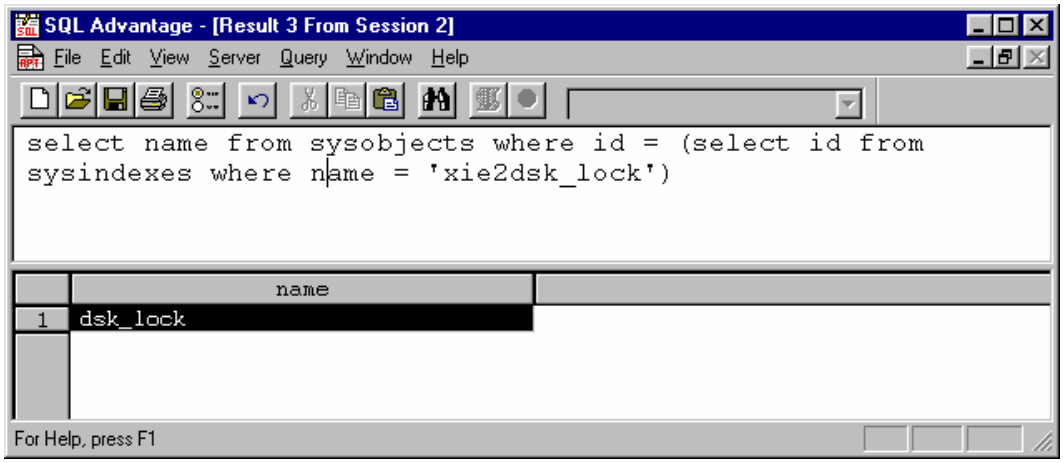
```
Index xie2dsk_lock is not consistent; found 72 leaf rows.  Drop
and recreate the index.
```

In this example, the name of the index that needs to be dropped and recreated is given. The following steps outline the process of recreating the index.

- **Step 1**: The name of the index is xie2dsk_lock. This is an index against the dsk_lock table since the naming convention of indexes in PD[2] generally includes the table name. However, you may want to verify this. Run the following query to verify the name of the table that an index belongs to when given the index name.

  ```
  select name from sysobjects where id = (select id from
  sysindexes where name = '<index_name>')
  ```

<index_name> = the name of the index from the error message.



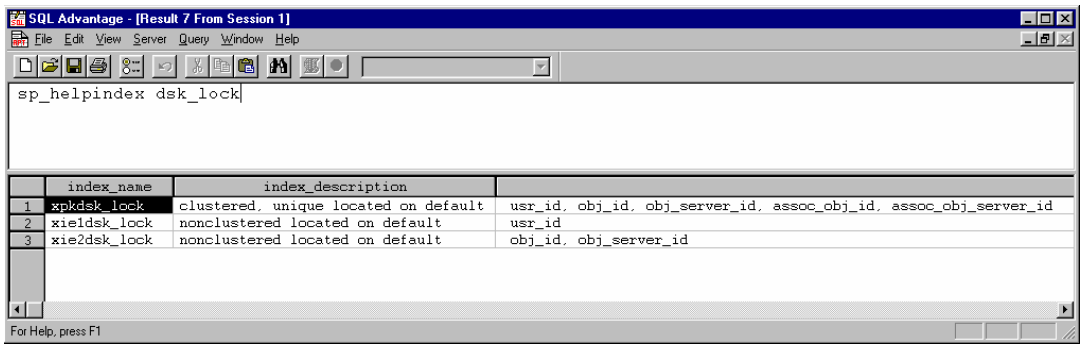
As you can see from the output above, the index 'xie2dsk_lock' is an index on the 'dsk_lock' table.

AMS Proprietary – Not for Disclosure Outside the Government

- **Step 2**: Before dropping the index, information needs to be gathered so that the index can be properly recreated. The following command will give you information about all indexes on a given table including the description (clustered, non-clustered, unique) and the index keys.

  sp_helpindex <table_name>

<table_name> name of the table that the index belongs to.



From these results, it can be seen that index xie2dsk_lock is non-clustered and has two index keys, obj_id and obj_server_id.

- **Step 3:** Drop the index. The syntax to drop an index is:

  drop index <table_name>.<index_name>

<table_name> = name of the table that the index belongs to
<index_name> = name of the index

Therefore, in this example, the command would be:

```
drop index dsk_lock.xie2dsk_lock
```

No results will be returned when this query completes executing.

- **Step 4**: Recreate the index.

The syntax to create an index is:

```
create [unique] [clustered | nonclustered] index index_name
on <table_name> (column [,…]) [with {{fillfactor |
max_rows_per_page} = # | ignore_dup_key |sorted_data |
[ignore_dup_row | allow_dup_row]}] [on segment_name]
```

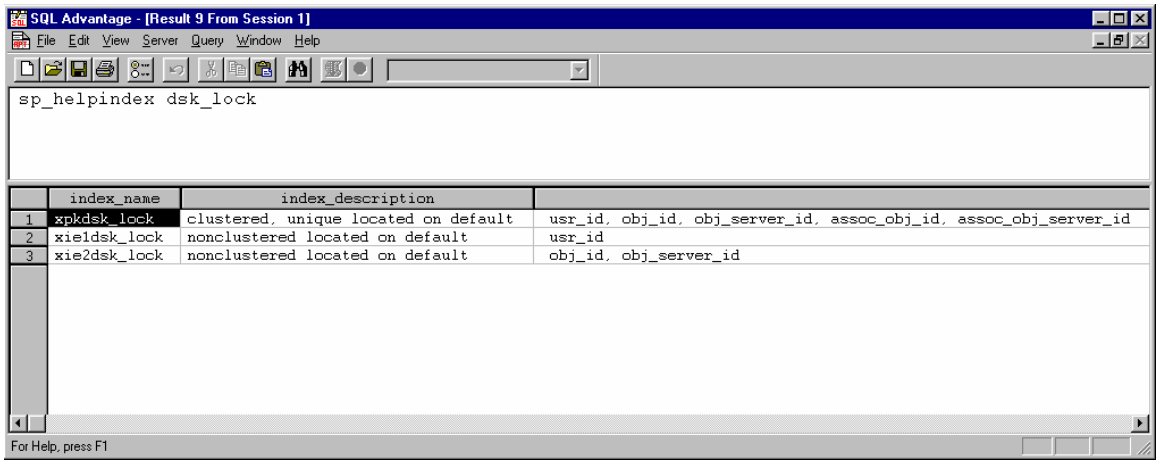
From this example, the command would be:

```
create nonclustered index xie2dsk_lock on dsk_lock (obj_id,
obj_server_id)
```

No results will be returned when the query completes executing.

AMS Proprietary – Not for Disclosure Outside the Government

- **Step 5**: Verify that the index was recreated by running **sp_helpindex dsk_lock**



	index_name	index_description	
1	xpkdsk_lock	clustered, unique located on default	usr_id, obj_id, obj_server_id, assoc_obj_id, assoc_obj_server_id
2	xie1dsk_lock	nonclustered located on default	usr_id
3	xie2dsk_lock	nonclustered located on default	obj_id, obj_server_id

- **Step 6:** Run update statistics on the table to update the page that contains information on how the data is stored in the table in relation to indexes.

  update statistics <table_name>

  <table_name> = name of the table that the index belongs to

From this example, the command would be:

update statistics dsk_lock

6.2 Example 2: Drop and Recreate Index (w/o Name)

In this example, you are not given the name of the index that needs to be dropped and recreated but you are given the table that the index is on as well as the index ID.

```
Key mismatch between index page 495335 (row 9) and data page
701957 in database 'U4PD2PP1'. Drop and re-create index id 2 of
table 'dsk_configuration'.
```

In order to find the name of the index, the following steps should be followed:

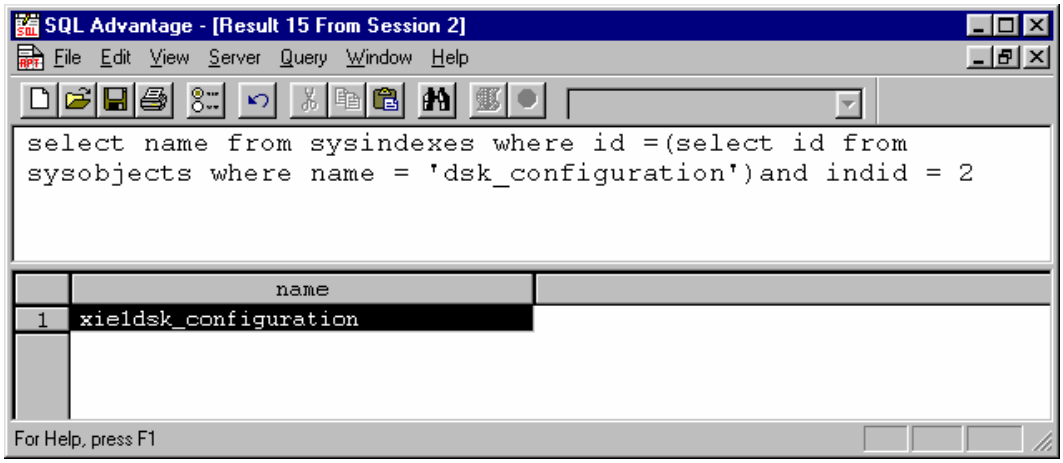
- **Step 1:** Determine the name of the index by running the following query:

  select name from sysindexes where id = (select id from sysobjects where name = '<table_name>') and indid = <index_id>

<table_name> = name of the table that the index belongs to as specified in the message

<index_id> = index id specified in the message.

AMS Proprietary – Not for Disclosure Outside the Government



From this result, we can see that the name of the index is xie1dsk_configuration. Now that we know the name of the index and the table that it is on, we can follow the steps from Example 1 to drop and recreate the index.

6.3 Rebuilding an Index Using Sybase Central

The SQL code needed to create an index can be generated using Sybase Central if the table and index name is known.

1. Log into the server using Sybase Central.
2. Open the "Databases' folder and click on your database name.
3. Open the 'User Tables' folder.
4. From the list of user tables, find the table for which the index needs to be rebuilt and double click on it.
5. Open the 'Indexes' folder. You should see a list of all indexes for that table.
6. Right click on the index that needs to be re-created and select 'Generate DDL'
7. The SQL statement for creating that index will appear, as seen below. After dropping the index in SQL Advantage, it can be recreated using the generated SQL statement.

AMS Proprietary – Not for Disclosure Outside the Government

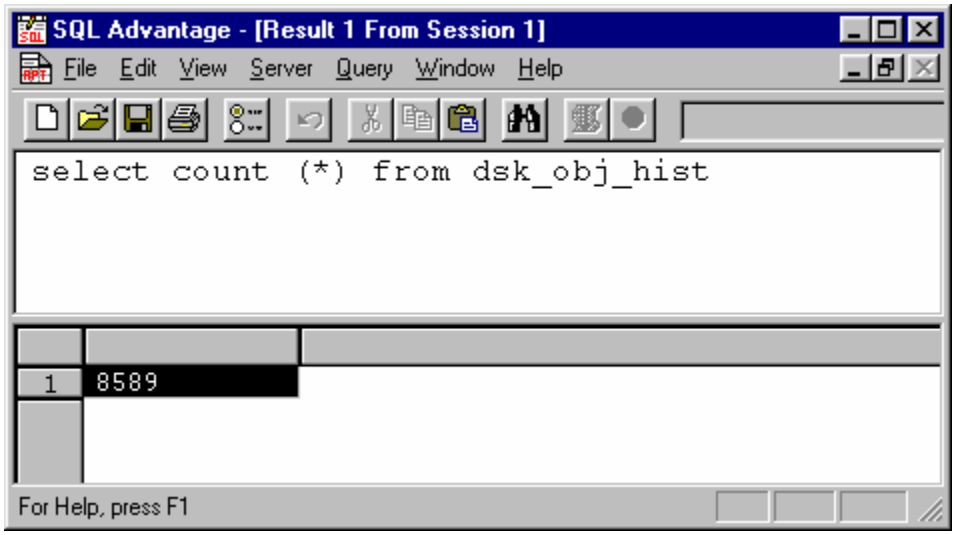
7. Rebuilding a Corrupt Table using 'select into'

Occasionally, a table needs to be rebuilt in order to remove corruption from it. In the following example, we will assume that the dsk_obj_hist table is corrupt and needs to be rebuilt

1. Determine how many rows are in the corrupt table by executing the following query:

   `select count (*) from <table_name>`

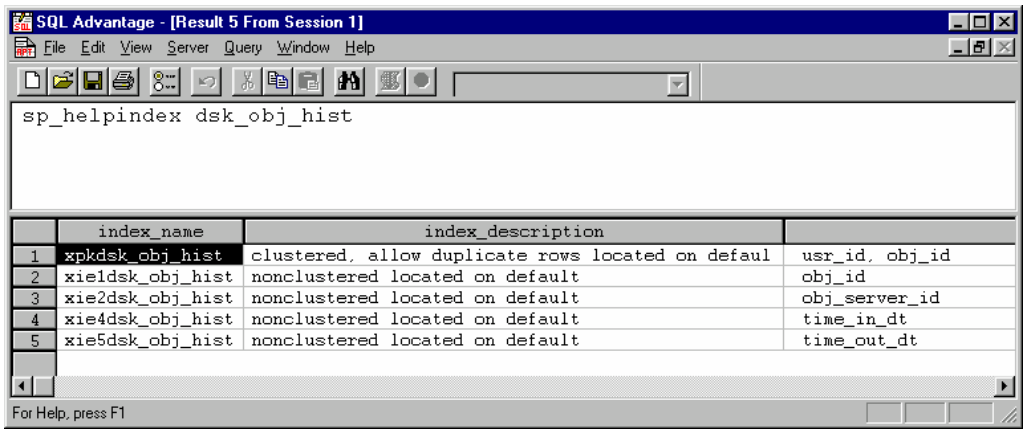
<table_name> = name of the table being rebuilt.



In this example, it can be seen that 8,589 rows exist in this table.

2. Run the following query to determine if the table has indexes and save the output.

   `sp_helpindex <table_name>`



	index_name	index_description	
1	xpkdsk_obj_hist	clustered, allow duplicate rows located on defaul	usr_id, obj_id
2	xie1dsk_obj_hist	nonclustered located on default	obj_id
3	xie2dsk_obj_hist	nonclustered located on default	obj_server_id
4	xie4dsk_obj_hist	nonclustered located on default	time_in_dt
5	xie5dsk_obj_hist	nonclustered located on default	time_out_dt

AMS Proprietary – Not for Disclosure Outside the Government

3. Run the following query to determine if the table has triggers and save the output.

```
select * from sysobjects where type = 'TR' and deltrig =
(select id from sysobjects where name = '<table_name>')
```

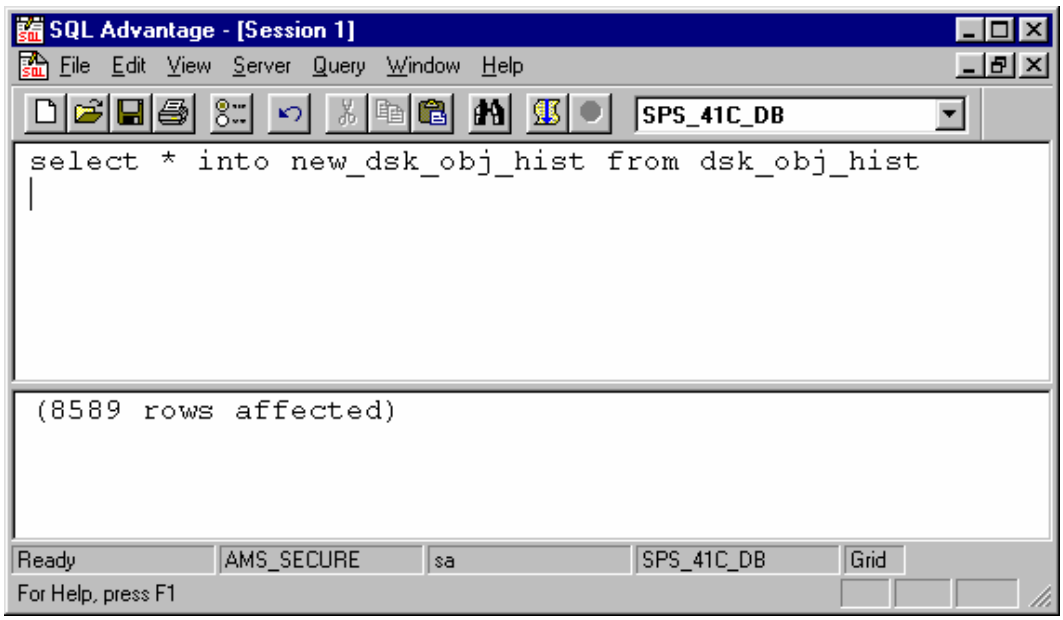
If the table does have triggers, make sure you obtain the SQL used to create the triggers by using the Generate DDL procedures outlined in Section 8.

4. Copy all of the data from the corrupt table into a new one by executing the following command:

```
select * into <new_table> from <table_name>
```

<new_table> = unique table name

<table_name> = name of the corrupt table

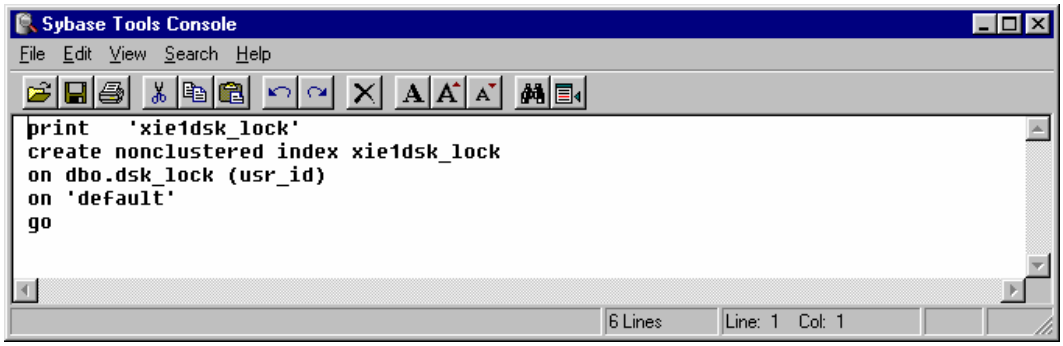


Make a note of the number of rows affected from the results window. It should match the number of rows from step 1.

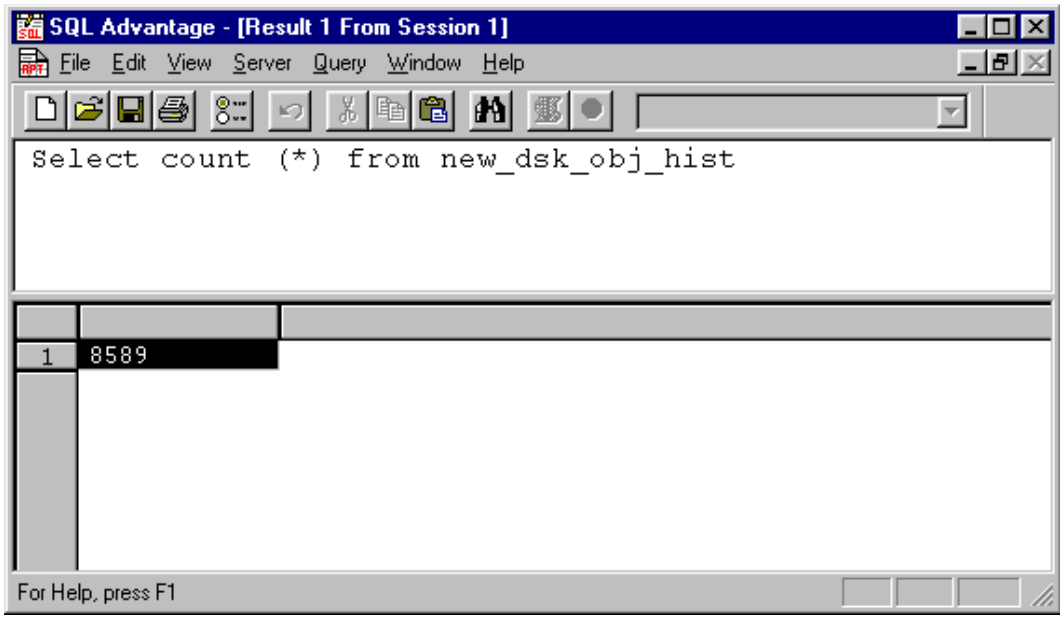
5. Verify the number of rows in the new table by executing the following command:

```
select count (*) from <new_table>
```

<new_table> is the table name from step 4.



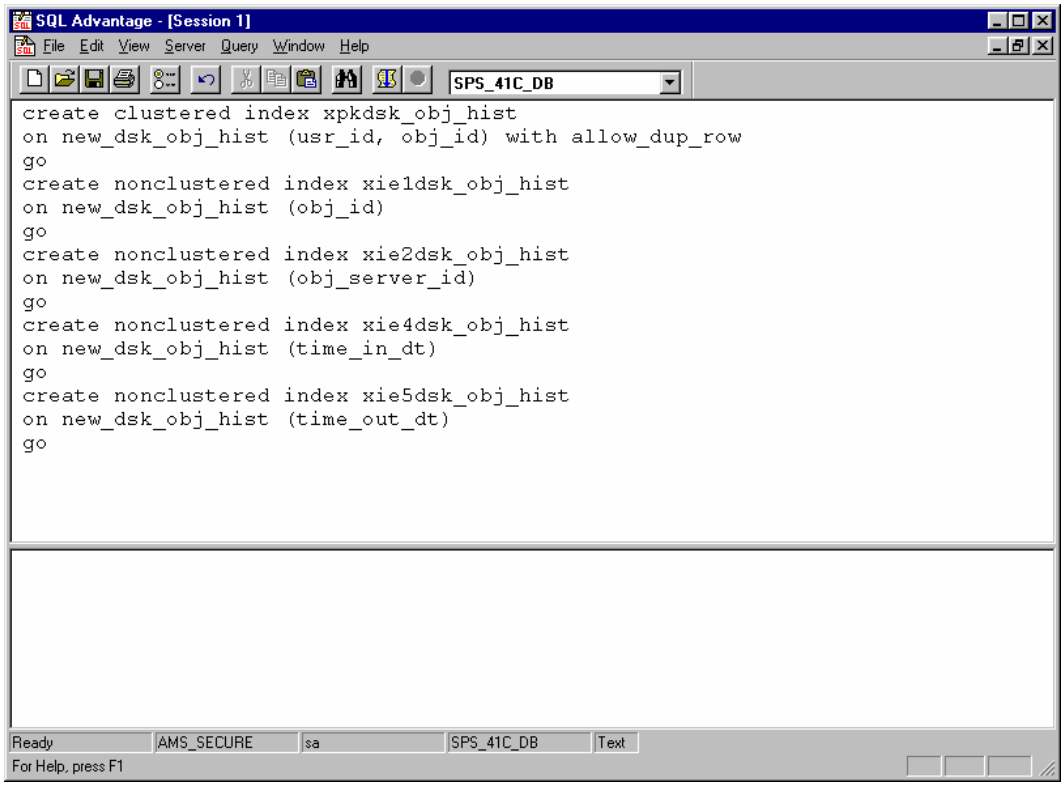
AMS Proprietary – Not for Disclosure Outside the Government



Once again, the number of rows returned should match the number from step 1.

6. Create indexes if necessary.

The following text is an example of the query necessary to create indexes for this example. For more information on creating indexes see Section 6.



AMS Proprietary – Not for Disclosure Outside the Government

7. Create the triggers, if necessary. See chapter 8 for details.
8. Run the following dbcc commands on the new table to verify that it is not corrupt:

   ```
   dbcc tablealloc (<new_table>)
   dbcc checktable (<new_table>)
   ```

   Example:

   ```
   dbcc checktable (new_dsk_obj_hist)
   dbcc tablealloc (new_dsk_obj_hist)
   ```

9. Drop the old table by executing the following command:

   ```
   drop table <table_name>
   ```

   Example:

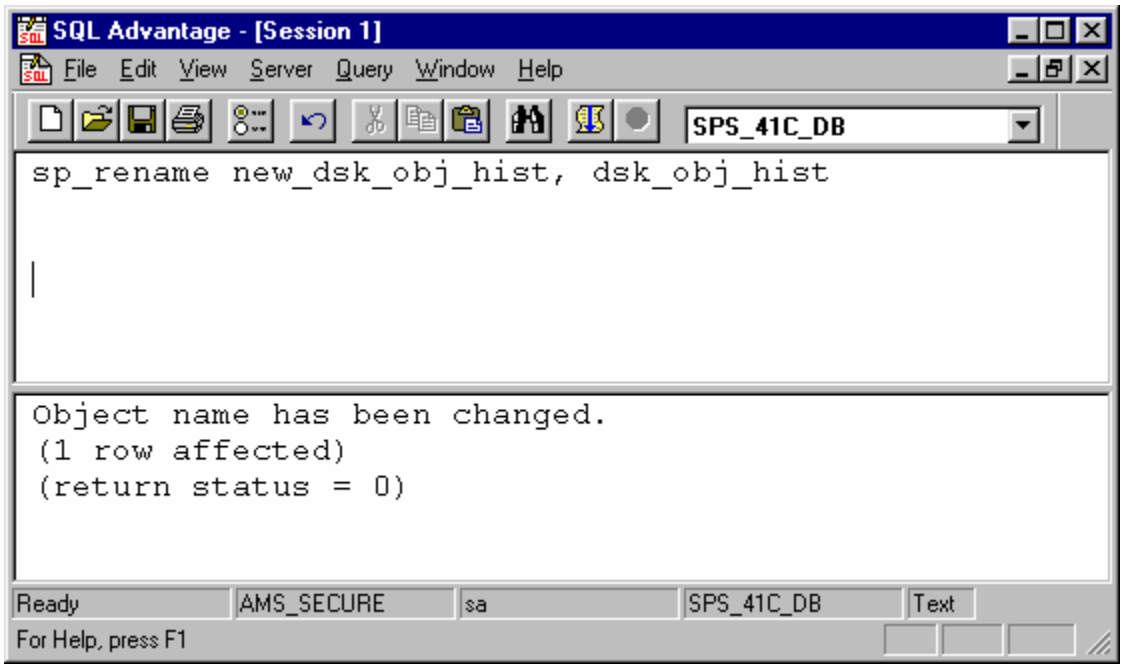
   ```
   drop table dsk_obj_hist
   ```

   No status messages will be returned.
10. Rename the new table to the name of the table that was just dropped

    ```
    sp_rename <new_table>, <old_table>
    ```

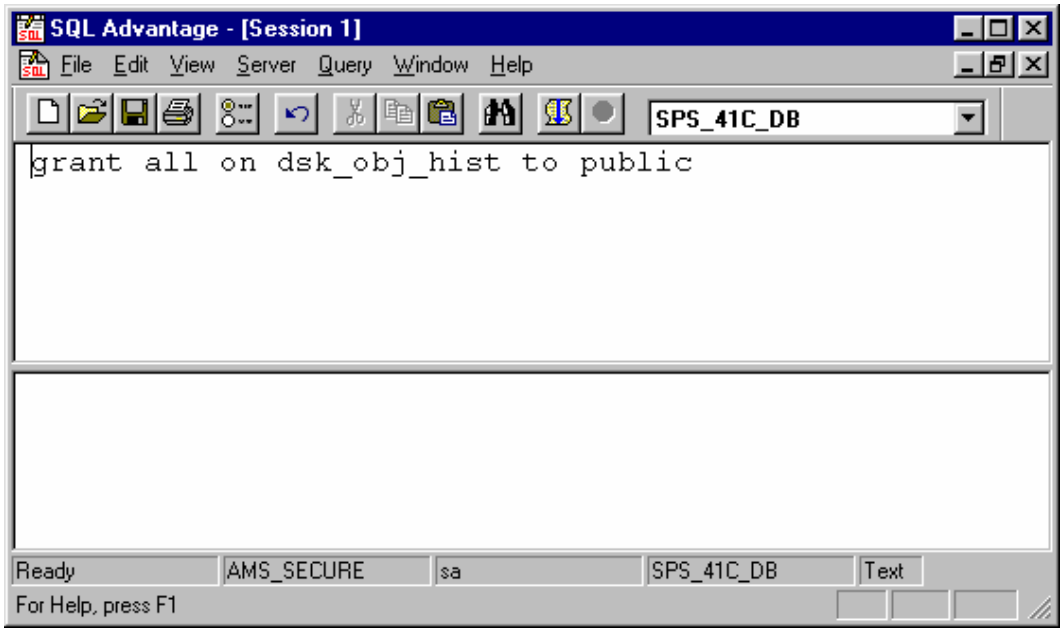

AMS Proprietary – Not for Disclosure Outside the Government

Example:



11. Add permissions to the table by executing the following command:

    grant all on <table_name> to public



12. Run update statistics on the table to update the statistics page. The statistics page contains information on how the data is stored in the table in relation to indexes.

    update statistics <table_name>

    Example:

    update statistics dsk_obj_hist

AMS Proprietary – Not for Disclosure Outside the Government

8. Trigers

When rebuilding a table, it is important to make sure that if the table has triggers, the triggers are also added to the table.

8.1 Verifying if a Table Has Triggers

8.1.1 Using SQL Advantage

Run the following SQL command to check if a table contains triggers

```
select * from sysobjects where type = 'TR' and deltrig =
(select id from sysobjects where name = '<table_name>')
```

<table_name> = name of the table being checked for triggers.

If the table has triggers, you can run the following command to view the SQL used to create the trigger.

```
sp_helptext <object_name>
```

<object_name> = name of the trigger.

8.1.2 Using Sybase Central

1. Within Sybase Central, log into the server and double click on the "Databases' folder.
2. Double click on the database name.
3. Open the 'User Tables' folder.
4. From the list of user tables, find the table that is being checked and double click on it.
5. Open the 'Triggers' folder. If the table has triggers, you will see a list of them.
6. To generate the SQL statement needed to rebuild the trigger, right click on the trigger and select 'Generate DDL', this should be done for each trigger.
7. Run the output from generating the DDL in SQL Advantage to recreate the triggers.

AMS Proprietary – Not for Disclosure Outside the Government