

# Data Recovery & Repair: A Practical Handbook

*A technical white paper by  
White Sands Technology, Inc.*

Last Updated: 5/5/2008 13:58:00



<http://www.whitesands.com/>

Copyright © 1996-2008 White Sands Technology, Inc. All rights reserved.

# Table of Contents

<b>Preface.....</b>	<b>1</b>
<b>Introduction – Disaster Strikes.....</b>	<b>1</b>
<b>Database Corruption Triage Procedure.....</b>	<b>2</b>
<b>A Corrupt Database is like a Crime Scene.....</b>	<b>3</b>
<b>Perform Non-Intrusive Checks.....</b>	<b>3</b>
Protect Your Database – make ‘NOFIX’ the default!.....	3
List of Safe DBCC Commands.....	4
<b>Make a Raw Binary Copy of the Database.....</b>	<b>4</b>
<b>Do Not Rebuild Tables or Indexes.....</b>	<b>5</b>
<b>Get Users Out of the Database.....</b>	<b>6</b>
<b>Summary.....</b>	<b>6</b>
<b>Understanding Data Corruption.....</b>	<b>7</b>
<b>What is Data Corruption?.....</b>	<b>7</b>
<b>How Does Corruption Occur?.....</b>	<b>7</b>
<b>Failures in Computer Hardware.....</b>	<b>8</b>
Disk Drives.....	8
Other Failures: CPU, RAM, Network.....	8
Power Failures.....	8
<b>Disk Drive Failures.....</b>	<b>9</b>
Database Logging.....	10
Disk Redundancy.....	11
Replication.....	11
<b>Failures in the Operating System.....</b>	<b>11</b>
<b>!@#&amp;% (Corruption) Happens, So You Need a Recovery Plan!.....</b>	<b>12</b>
<b>Detecting Database Corruption.....</b>	<b>12</b>
<b>Run Regular DBCCs—But Be Careful!.....</b>	<b>12</b>

Improving on DBCC: Third-Party Solutions.....	14
Finding Data Corruption the Hard Way (From Users).....	14
<b>When !@#&amp;% (Corruption) Happens.....</b>	<b>14</b>
Investigating the Cause of the Corruption.....	15
Determining the Extent of the Corruption.....	16
<b>Standard Disaster Recovery Strategies: Benefits, Drawbacks and</b>	
<b>Limitations.....</b>	<b>17</b>
Restart the Database Server.....	17
Copy Out and Reload the Data.....	17
Drop and Rebuild the Corrupt Object.....	18
Switch In Your Mirrored Disk.....	18
Restore From a Backup.....	18
Restore From a Replicated Database.....	19
Use DBCC to Repair the Corruption.....	19
<b>Advanced Data Recovery Techniques.....</b>	<b>20</b>
<b>General Corruption Repair Strategies.....</b>	<b>20</b>
<b>Repair the Data-Layer Pages.....</b>	<b>21</b>
Repairing APL Tables.....	21
Dropping a Page From the Page Chain.....	21
Repairing DOL Tables.....	22
Repairing Allocation Errors.....	23
Repairing a Corrupt Index.....	24
<b>Recovering Damaged Data.....</b>	<b>24</b>
<b>Understanding Sybase Error Messages.....</b>	<b>26</b>
About Sybase Error Logs.....	26
<b>Error 605.....</b>	<b>26</b>
Diagnosing 605 Errors.....	27
Repairing 605 Errors.....	28
<b>Error 692.....</b>	<b>28</b>
Diagnosing 692 Errors.....	29
Repairing 692 Errors.....	29
<b>Error 695.....</b>	<b>30</b>

Diagnosing 695 Errors.....	31
Repairing 695 Errors .....	31
<b>Error 697 .....</b>	<b>32</b>
Diagnosing 697 Errors.....	32
Repairing 697 Errors .....	33
<b>Error 806 .....</b>	<b>33</b>
Diagnosing 806 Errors.....	33
Repairing 806 Errors .....	34
<b>Error 1133 .....</b>	<b>34</b>
Diagnosing 1133 Errors.....	35
Repairing 1133 Errors .....	35
<b>Conclusion .....</b>	<b>36</b>
<b>Index .....</b>	<b>37</b>

---

## Preface

This white paper is intended for DBAs and managers who must be concerned with keeping production database systems running and error-free. It assumes some familiarity with database servers (particularly Sybase Adaptive Server Enterprise).

While the focus of many of the details in this document is on Sybase database servers, the general principles apply to all major RDBMSs.

---

## Introduction – Disaster Strikes

*Emergency!*

It's late Friday afternoon. Most businesses on the U.S. West Coast are wrapping up their week's work, and the rest of the country has already started heading out for happy hour.

At 3:30pm PST, a support technician at White Sands Technology, Inc. receives a call from a frantic Bob, the senior DBA at a major Wall Street financial services company<sup>1</sup>. Their Sybase online trading database server is logging cryptic errors that report an incorrect object ID on a page in one of their mission-critical web-access account tables, and certain users are getting knocked out of the system.

Since the database server is replicated to a warm standby server, Bob naturally checked the standby server and found that it too had data corruption, but in different places from the production server. So there is no possibility of restoring from the standby server.

Bob has also tried loading last month's backups onto their smaller test server, to see if the corruption occurred recently. To his dismay, he determined that the index corruption has existed for at least 30 days, undetected until now.

They cannot simply reload a backup older than 30 days and reapply the changes made since then, because the time required to do that would bring their whole trading system down for the better part of a day, thus costing the company millions of dollars in lost orders.

They cannot simply run a DBCC (database consistency check) command to fix the table, because DBCC could truncate the table at the point where the page chain corruption exists, thus dropping thousands of rows from the table.

It seems to Bob that this is a no-win situation.

However, the White Sands technician has a much different take on it.

She presents Bob with the following options, which are made possible by White Sands's Disaster Recovery Toolset:

---

<sup>1</sup> All names and some identifying details of this story have been changed to protect the anonymity of the client.

- Use the Disaster Recovery Toolset to scan the database, identify **all** the pages belonging to the corrupt table—even pages inaccessible to Sybase—and export the rows on all the pages to an ISQL script in a flat file, so the data can be reloaded later.
- Use the Disaster Recovery Toolset's Database Page Editor to repair the corruption on-line.

Since minimizing downtime is Bob's paramount concern, the second option, while more labor-intensive, works much better for him.

*Diagnosing the problem.*

Over the next couple of hours the White Sands technician works with Bob to dig deeper into the problem. Using the Disaster Recovery Toolset from White Sands, we determine that the table's page chain is corrupt in one area of the table—specifically, the data page chain is broken. The Disaster Recovery Toolset quickly identifies the page where the page chain break existed, and also identifies the most likely page to which the corrupt page should have been joined.

Further checking identifies a nonclustered index row that points to the invalid page. This and the broken page chain appear to be the entire extent of the damage.

*The Happy Ending!*

How did everything turn out? Well, fortunately the story has a happy ending and, of course, a moral. Using the Disaster Recovery Toolset, the technician was able to repair the damaged pages while keeping the database online. The fix was made first on their standby server, so they could verify that the repairs would work correctly. After testing the fix on the standby server, a similar fix was done on the production server. No data was lost, and no downtime was incurred.

The moral is that while backups and replicated systems are an important part of your data integrity assurance plan, these methods alone are not enough to ensure zero downtime and zero loss of data.

So how was this seeming miracle worked? Let us take a closer look.

---

## Database Corruption Triage Procedure

This section is intended as a guide to what do do—and what **not** to do—when you encounter corruption in one of your databases.

When a database corruption situation occurs, unfortunately many DBAs will find themselves unprepared to deal with the emergency. To our knowledge, database vendors do not have front-line support personnel who have special experience in dealing with data corruption, so you may have to wait through an escalation procedure in order to get in touch with qualified technicians who can help you deal with corruption.

Or, you may turn to fellow DBAs who have had experience with corruption, or do your own research on the Internet in order to gain further insight into the corruption issue you are facing.

The problem we find with these methods is that you can get incomplete or downright inaccurate advice, for the simple reason that virtually no one in the database industry has spent a significant amount of time developing solutions to database corruption.

We at White Sands have tried to address this "information gap" not only by building the world's best tools for diagnosing and repairing data corruption, but also by sharing the expertise we have gained through working on innumerable database corruption cases.

## A Corrupt Database is like a Crime Scene

Treat a corrupt database as a police detective treats a crime scene: don't let anyone touch anything!

Any action taken which modifies the state of the database, can result in **additional data loss** and can obfuscate the cause of corruption. If you can take the database offline without adversely affecting your organization, this is recommended. Disallow others from running any commands to diagnose or attempt repair of the corruption.

## Single Point of Control

During a data corruption incident, **all** recovery, repair and salvaging should be centralized through one person to minimize mistakes. Additionally, **all** commands that are executed against the server should be documented and **all** output should be saved in its entirety. This information may prove to be useful later in determining the extent or the cause of corruption. It is always best to work from a backup of the database. However, keep in mind that a Replicated server might also suffer from the same or other data corruption. The best way to make a backup is to make a raw binary copy, preferably directly onto another machine besides the one having corruption problems (more on this later).

## Perform Non-Intrusive Checks

*DBCC can be dangerous  
to your data!*

When corruption is discovered, the instinct of many DBAs is simply to "let DBCC deal with it." DBCC (the Database Consistency Checker command in Sybase) can check and, in some cases, repair the internal consistency and storage structure of a database.

However, using DBCCs on a production database must be done carefully, for the following reasons:

- The procedure by which DBCC "fixes" corruption will sometimes cause **loss of data**
- For some DBCC commands (i.e. tablealloc, indexalloc, textalloc), the "fix" option is **on by default** for user tables
- When DBCC "fixes" corruption, it **does not log the changes it makes**, and it **does not make a backup** of the previous contents of the pages it modifies; thus, its changes **cannot be undone**

**In a corruption situation, we strongly recommend that you use only the non-intrusive ("nofix") forms of DBCC commands on your production database. Load a copy of an "exact" binary dump onto a test server and run the "fix" option there.**

## Protect Your Database – make 'NOFIX' the default!

For some DBCC caommands (i.e. tablealloc, indexalloc, textalloc), **it is best to change the default option to "nofix". This can be done by enabling the following boot trace flag: T2507 in your database's startup script.**

## List of Safe DBCC Commands

Here are some of the DBCC commands which are **safe** to run in a corruption situation:

- dbcc tablealloc(*table\_name*, full, NOFIX)
- dbcc textalloc(*table\_name*, full, NOFIX)
- dbcc checkalloc(*database\_name*, NOFIX)
- dbcc checktable(*table\_name*)

This allows you to determine the full extent of the corruption before taking any action which might modify the database.

*Save all DBCC output!*

Here is another step which we find is often omitted by DBAs: **always save the output of all DBCC and other commands** you run. All too often, we encounter situations where DBCCs were executed by an automated ("cron"-type) script, and the output was discarded or selectively saved based on greps for error messages. This makes it impossible to tell whether, or where, DBCC has made changes to the database.

Even if you "grep" the DBCC output for errors or corruption messages, you still should save the original output in case there were other messages that could later prove to be useful.

In all your automated scripts (where hopefully you will run DBCC with the "nofix" options as described above), you should save the full DBCC output for a period of time (e.g. one year) in case you later discover corruption in a database and need to go back and see what DBCC found.

## Make an "Exact" Binary Copy of the Database

*Dumping a corrupt database may produce a corrupt or incomplete backup!*

When you have discovered corruption in a database, it is a good idea to make a backup copy of the database before taking any action that might modify the database. In addition to having a backup for safekeeping, you can load the backup onto another server and test any proposed fixes to make sure they will work in production.

But note—if corruption already exists in your database, you should **not** rely on the standard dump-and-load method for backing up the database!

**In Sybase ASE, The DUMP command is in optimized mode by default. Optimized dumps are designed to make backups smaller by skipping unused portions of the database. This type of backup is not an "exact" binary copy of the database.**

More specifically, here are some of the ways in which an optimized DUMP command might not make an exact backup of the corrupt database:

- It can fail to copy incorrectly allocated pages, thus causing some data to be missing when you restore the database;
- If the above occurs, the restored database can retain some of its old contents from prior to doing the restore, thus covering up actual corruption that exists in the original corrupt database;
- The DUMP command may fail altogether if certain types of corruption exist.

*How to make a valid copy of a corrupt database.*

Your best bet is to make an exact binary copy of the database devices, preferably directly onto another machine, using the following methods, as applicable:

- **Recommended.** In newer versions of Sybase ASE (starting around 12.0.x) the `sp_dumpoptimize` command's "maximum" option can be used to attempt to dump all pages of a database, whether allocated or not. We do not recommend that you rely solely on this, however, since it may still fail to copy corrupt pages from the source database.

The following command will enable this option:

```
sp_dumpoptimize 'archive_space=maximum'
```

To disable this option:

```
sp_dumpoptimize 'archive_space=default'
```

- On Unix operating systems, use the 'dd' command to copy raw disk devices, partitions or logical volumes to a backup location
- On Unix, you can use the 'cp' command to copy filesystem-based device files to an alternate location
- On Windows, the 'copy' command will let you copy file-based device files to a backup location

Note also that before using one of the above methods, you should checkpoint the database, and preferably take it offline or place it in single-user mode, so as to avoid or at least minimize the possibility of having any changes made to the database in memory (cache) that have not been written to disk when you copy the database.

Of course, if you have any corruption at the operating system level or on the physical disks, then you may not be able to make a valid copy of the database at all. But doing an operating system-level backup will get you as close as possible to an exact copy of your corrupt database.

## **Do Not Rebuild Tables or Indexes**

It is often suggested to rebuild an index or an entire table in order to eliminate the corruption from your database.

We believe this is a very bad idea for two reasons:

- In dropping an index or table, you can inadvertently drop (deallocate) pages that actually belong to another object in the database;
- When you drop and rebuild a corrupted index or table, you can inadvertently overwrite data pages belonging to other objects in the database, thus **causing data loss.**

*Use a test server to try dropping or rebuilding objects—not the production server!*

If you want to try dropping or rebuilding objects to see if it will fix the corruption, you should only do this on a separate database (preferably on a different database server), and only on an **exact binary copy** of the production database, as described in the previous section. The recommended procedure is as follows:

1. Make an exact binary copy of the database onto a test or backup location.

2. Run full DBCCs (on all tables in the database) using the applicable "nofix" options, on the copy of the database.
3. Get a row count on each object involved in the corruption; use the force-index syntax to get a separate row count using each index in each affected table.
4. Drop or rebuild the objects you wish.
5. Run full DBCCs (again on all tables in the database), still with the "nofix" options where applicable.
6. Compare the DBCC output before and after dropping or rebuilding the objects, to see if the original corruption was repaired and/or any new corruption was caused.
7. Compare the row counts before and after the drop/rebuild, to see if any rows were lost during the operation.

The above steps are **not** a guarantee that the same fix will work on your production database, since data may have since changed on production. But it is still better than blindly trying out an attempted fix in production, only to find out later that the "fix" actually made the corruption worse.

### **Get Users Out of the Database**

This step is probably the most difficult in a real-world corruption situation on a live production database. The typical large-enterprise database system is not just sitting idle all day, it is being used by hundreds or thousands of users.

When corruption has occurred in a database, making more changes to the data runs the risk of making the corruption worse or causing further data loss. Thus, we recommend, to the extent possible, limiting users' ability to make changes to the database.

### **Summary**

We hope this section has provided helpful information regarding how to handle a data corruption incident, should one occur in your enterprise.

The next section delves deeper into the causes of data corruption.

---

# Understanding Data Corruption

Before we can discuss how to fix data corruption in your databases, we must first explain exactly what data corruption is and how it occurs.

## What is Data Corruption?

In a nutshell, data corruption (which we also refer to as *database* corruption in this white paper) is any error in the data storage structure used by your database.

Sybase databases have a well-defined storage structure, which consists of *database pages* at the lowest level of granularity, and has higher-level structures based on blocks of pages within the database. Each database page is 2K in size (or larger in newer server versions).<sup>2</sup>

Within each database page, data is stored in a well-defined format which depends on the type of page (data, index or other page types).

*Data corruption* occurs when the formatting of one or more pages becomes damaged, often rendering information on that page (and/or other pages) inaccessible to the database server.

*Transient vs. persistent corruption.*

Database corruption may be *transient*, meaning the corruption only exists in the cached copy of data which is resident in the server's RAM memory; or, it may be *persistent*, meaning the corrupted data has been permanently written to the server's hard disk(s).

As a result of corruption, the database server (and thus your users) may encounter errors any time it tries to access a corrupt table at all, or it may get errors only when accessing certain rows or pages in the table, or only when accessing the table through a certain index.

Whatever the case, when data corruption occurs, it can prevent access to large amounts of your organization's data by your users and applications, and thus can obviously have a significant impact on your operations!

## How Does Corruption Occur?

There are several hardware and software elements which comprise your database server:

- The server's CPU, RAM and associated hardware (including power source)
- The disk controllers and drives, which are also hardware but warrant special consideration due to their central role in the database server.
- The operating system (Windows or Unix "flavor")

---

<sup>2</sup> For a detailed description of the exact data structure of Sybase databases, see the references on page 26.

If any of these elements fail, data corruption can result. Let us take a closer look at how each of these elements can fail, and how these failures can cause data corruption.

## **Failures in Computer Hardware**

Today's enterprise-class server machines are extremely reliable, to a degree that might have been unfathomable ten or twenty years ago.

But, as reliable as computer hardware is today, it still does fail occasionally. Fortunately, when a failure occurs, it is usually in an obvious way, so that errors do not go undetected.

This is fortunate, because when a hardware error or failure occurs, you will want to know about it as soon as possible, and with the least possibility of incorrect information getting stored into the computer permanently.

To see why undetected failures are so rare in computers today, let us examine the different components in the computer which can fail.

### **Disk Drives**

By far the most common hardware component to fail in a computer is the hard disk drive, since it has the most moving parts (and also the fastest-moving parts). Since disk drives are a uniquely integral part of a database server, very special measures are taken to deal with their failures, which we will discuss in detail in the section starting on page 9.

### **Other Failures: CPU, RAM, Network**

Aside from disk drives, most of the components in a computer are solid-state (meaning they are entirely electronic and have no moving parts).

If operated within the proper ranges of temperature, humidity and shock (physical vibration), these components will typically run for decades without any failures, and will require little or no maintenance.

For the rare occasions when electronic circuitry does fail, built-in error detection mechanisms are used so that the hardware and/or the computer's operating system can detect the failure and deal with it accordingly.

The result is that systems today generally handle these situations appropriately so that the computer can proceed to a known state whenever a critical hardware failure occurs.

### **Power Failures**

System power failures are a type of hardware failure that is common enough to be planned for by system hardware designers.

*A UPS is an essential part of a high-availability system.*

Uninterruptible power supplies (UPSs) are typically used in high-availability environments to ensure that short power outages will not affect the servicing of end-users.

Two other database system facilities also help ensure that data integrity will be maintained in the event of a power failure:

- UPSs typically provide a "low battery" signal to the computer system, so that if the UPSs battery will soon run out, the operating system, database server and other applications running on the machine can begin an orderly

shutdown so that all transactions in progress will either be rolled back or will get saved to disk correctly.

- The database's transaction log (described more fully in the next section) helps ensure data integrity, even if power is removed from the computer suddenly while transactions are in progress.

## Disk Drive Failures

Disk failures are given special consideration by typical enterprise-scale operating systems and database servers. This is done for two reasons:

- First, because disk failures are probably at least an order of magnitude more common than failures in any other critical component of a computer system;
- Second, because undetected errors or inconsistencies in the information stored on a computer's disk drives can compromise the integrity of a company's critical business data, thus potentially jeopardizing the company's entire business as well as the businesses of its customers.

Let's discuss these two points in more detail.

*It's not whether your disks will fail, it's when.*

Hard disk drives today have a *mean time between failure* (MTBF) ranging from 300,000 to over one million hours, and a *service life* of 5 years or more. Since the MTBF is so much greater than the service life, this means that, on average, a high percentage of disk drives will run for a period of time at least as long as the service life without an *unrecoverable* failure.<sup>3</sup>

An unrecoverable failure means an error or failure in writing or reading data to or from the disk, such that the error remained after all available *error correction* methods were tried.

In actuality, disk failures can start to occur much sooner than the MTBF, as we will discuss below; also, keep in mind that the MTBF is an average, meaning that some disk drives will fail much sooner than the manufacturer's specified MTBF. (True, some drives will also last much longer than the MTBF, but you should be a lot more concerned about the ones that fail sooner!)

*Error detection.*

Before you (or a disk drive) can try to correct any errors that might occur, you first need to know that an error has in fact occurred. To do this, disk drives employ various *error detection* techniques, such as checksums, cyclic redundancy checks (CRCs) and other algorithms. Methods used today can detect at least 99.99% of all unintended modifications to data on the disk.

*Error correction.*

Once a disk error has been detected, there are a number of strategies typically employed to try to correct the error.

Error correction methods include retrying the read or write operation 5 or 10 times; they might also include application of *redundant error correction data* to the erroneous data to see if the correct data can be restored.

The upshot of all this is that, depending on the effectiveness of these error-correcting measures, a disk might start to experience failures long before the

---

<sup>3</sup> See Appendix C in <http://www.redbooks.ibm.com/redbooks/pdfs/sg244551.pdf> for more information on understanding MTBF.

average failure time of 2 to 10 years, but these failures go undetected (and in fact are not significant—yet) because they were automatically corrected by the disk drive's hardware, circuitry and/or embedded firmware.

It is when these errors can no longer be corrected by the disk drive itself that they start to have a real impact. Depending on what type of failure occurred in the disk drive, you might see different symptoms:

- You might see errors reported by your database server or operating system very infrequently, and the errors slowly and gradually become more and more prevalent;
- You might see errors only in certain locations of the disk (i.e. when accessing certain files, tables, etc.)
- In the worst case, an entire disk might “die” suddenly, taking all of its data with it to that “big databank in the sky.”

Today's enterprise-scale database server systems (and the people who design, install and configure them) employ various methods to prevent loss of your organization's data in the event of any of the above types of disk failures.

### Database Logging

As any experienced DBA knows, database servers such as Sybase use a logging mechanism (such as the *transaction log* in Sybase), one of the major benefits of which is *recoverability* of the data stored in the database—if one or more of the database's data or log disks should fail, the data can be reconstructed to a consistent state from the last backup combined with the transaction log data since the last backup.

*Why use transaction logging?*

Maintaining a disk-based transaction log, and using it to correctly update the data in a database, is an enormously complex task. Why, then, do database servers go to all this trouble? Why not just keep a record of transactions in RAM, and finish updating data on the disks when transactions are completed (committed)?

The answer is simple: because sooner or later, disks are all but guaranteed to fail. And when they do, the database server does its best to prevent any loss of data, or loss of integrity in the data stored in the database. The transaction log provides a form of *disk redundancy*, in that if either the data device or log device (but not both) in a given database should fail, the server can recover the database to a known, transactionally consistent state automatically.

*Backing up transaction logs is very important!*

Incidentally, this is a good argument for **not** enabling the Sybase “truncate log on checkpoint” option on any production databases. If you use this option, in the event that corruption occurs, you will be unable to recover the database from backups to a point in time close to when the corruption occurred. The same argument holds in the event of a disk failure.

By performing regular database dumps to tape, and performing frequent transaction log dumps to tape in between the database dumps, you ensure that you can recover the database up to that point in time when the failure occurred.

The point is that, unless you can stand to lose the changes made to the database since the database dump occurred, database dumps alone are not enough—you need the transaction log backups also.

## Disk Redundancy

Today's operating systems, enterprise-class database servers and disk storage subsystems all can provide various options for providing redundant disk storage. This means that every block of data (on a disk, a database or an entire system) is stored two or more times, on different disks (and possibly accessed through different controllers or entirely different subsystems). Thus, should a single hardware failure occur in any of the disks, its redundant backup disk will take over, and in theory no data loss should occur.

## Replication

Many database servers including Sybase offer the option to *replicate*, or maintain a copy of, databases on a completely separate system. As data is added, changed or deleted on the main, or *primary*, server, these changes are sent to the replicated server and mirrored there, so that an exact logical copy (typically within a lag time of a few minutes or less) is maintained on the replicated system.

This has the advantage of full redundancy—every component of the database system (CPU, disks, network cards, etc.) is backed up by the replicated system.

There are other potential benefits to using replication, depending on the situation—you can split users between the two systems for load-balancing; or, you can keep one server offline to users as a “warm standby” and perform maintenance on it while users use the primary system, for example. After performing the maintenance on the standby system, you would then switch the standby into production, and switch the former production server into standby mode.

## Failures in the Operating System

Although a computer's operating system is no more than a specialized type of software program, it is unique in that all application software running on the computer depends on the operating system for various classes of services such as allocating system resources, disk I/O, communications, etc.

Because the OS supports all the other software running on the computer, it tends to “disappear into the background” in the eyes of users (and even system administrators) and become part of the machine itself.

Also because of this, the OS is usually the most stable, reliable piece of software on the computer—almost as reliable as the computer hardware itself.

These two factors conspire to make the OS seem “above reproach”—when a failure occurs, it often does not occur to anyone to investigate the operating system itself.

*Like any other software, operating systems can have bugs.*

However, experienced DBAs and system administrators know that this is not true at all—the OS, like any other highly evolved and complex piece of software, is not perfect and does contain bugs which can surface from time to time.

Depending on the seriousness of bugs which may occur in the OS, database corruption can in rare cases occur, and database integrity can be compromised as a result.

## !@#&% (Corruption) Happens, So You Need a Recovery Plan!

It might seem, with all these protective measures system designers and administrators take to ensure data integrity, that data corruption would be all but unheard of today.

Unfortunately, this is not true—despite all these measures, in certain cases, failures in the hardware, software or a combination of both *do* occur, and data corruption results. If and when it does, you will want to make sure you have put a good contingency plan in place beforehand.

*Backups alone are not enough.*

Merely using backups, or even replication, is not good enough to ensure that data corruption will have a minimal or no impact on your production servers. This is true for two reasons:

- 1) Because when corruption occurs, the corruption might not be detected when backups are performed, and thus the corruption will be silently propagated to the backup systems, meaning that you no longer have a completely good copy of your data anywhere; and
- 2) Because even if you have a good copy of the data (e.g., in a replicated database), getting it back on-line and into production may require more downtime than your operation can afford.

As we will see later, further measures must be taken to deal with corruption if and when it occurs, so that minimal downtime and loss of data occur.

*Fixing corruption with DBCC: Just Say No!*

We talk to many people who say, "If corruption occurs, why not just let DBCC fix it?" As we described in more detail in the Database Corruption Triage section earlier, when you use DBCC's "fix" options on a corrupt table, it sometimes fixes the corruption by truncating the table at the point of corruption, and/or deallocating the corrupt pages it finds.

This can add up to some dataloss. This may be acceptable in certain cases, but in the majority of instances, your business cannot sustain that kind of data loss. This was one of the major reasons White Sands developed its Disaster Recovery Toolset.

---

## Detecting Database Corruption

Now that we have discussed what database corruption is and how it occurs, how do you find out whether corruption exists in your databases?

### Run Regular DBCCs—But Be Careful!

As most DBAs know, the primary method for detecting database corruption is running DBCCs (database consistency checks) against each production database on a regular basis. It is a good idea to run DBCC checks as often as reasonably possible (e.g. once a day).

However, it is **not** enough simply to let DBCC run with the default options and consider the job done.

The main problem with using the default DBCC options to check for database corruption is that certain DBCC commands will automatically attempt to "fix" any corruption they find.

Furthermore, when DBCC truncates data from a table like this, the changes are **not logged** in the transaction log, and generally cannot be recovered. If this happens, only a tool such as White Sands's Disaster Recovery Toolset which directly reads the database devices, stands any chance of recovering the lost data.

Thus, by using DBCC with the wrong options, you can actually make corruption **worse** from the standpoint that you can lose data permanently, or at least make it more difficult to recover.

On page 3 we provide a list of some basic DBCC commands that are safe to run in a production environment.

*Running DBCCs should be viewed as a requirement.*

Running regular DBCC checks is so important that we view it as virtually a requirement, yet we encounter many production environments where this essential step is not being performed.

In many situations, especially 24x7 environments that require very limited database downtime, special steps must be taken in order to perform DBCC checking against production databases. Generally, DBAs will make a dump of the production database, load it onto a backup server and run the DBCCs there, this has two benefits:

- The DBCCs (which are disk I/O-intensive) do not impact the performance of the production server;
- The DBCCs execute reliably and without generating spurious or transient error or warning messages, since no users are making data modifications to the backup server while the DBCCs run (in fact, ideally each of the backup server's databases should be placed into single-user mode as the DBCCs run against it)

In replicated environments where a warm standby server is used, the DBA can take the standby system offline to run DBCCs while users use the primary system; then, after the DBCCs complete, the warm standby can be brought back on-line and caught up with the primary.

*Risks of not running regular DBCCs.*

Because of the extra work that must be done to perform DBCCs in a high-availability environment (and the extra hardware that may be required), many DBAs forego them entirely.

This is unfortunate, because data corruption often works its way into your databases in such a way that it does not immediately become noticeable or affect users.

If you do not perform DBCCs at least on a daily or weekly basis, your database backups may become worthless because they will contain corrupted data. Then, when the corruption surfaces, your backups may be useless since the backups for the last few weeks or more may all contain the corruption as well.

Another important point to mention is that you should run DBCC checks against your production *and* backup systems, since corruption can arise in one or the other independently. This is also true in replicated environments—corruption can occur in the replicated system that does not exist in the primary system, and vice-versa.

## Improving on DBCC: Third-Party Solutions

A few third-party software vendors, including White Sands Technology, Inc., offer database verification products that complement the use of DBCCs and help the DBA identify database corruption.

The benefits of these third-party solutions can include the following:

- Detect and reporting of certain kinds of corruption that DBCC does not;
- Ability to operate while the database is on-line (i.e. without having to lock the database or put it into single-user mode);
- They typically report corruption-related errors and information in a much more concise and easy-to-understand format than the DBCC output.

Keep in mind that these solutions should be used *in conjunction with* DBCCs, not instead of them! The more checks you run against your databases, the more likely you are to detect corruption as soon as it occurs. Furthermore, most third-party tools do not provide as thorough checking of your database as DBCC does.

So we strongly recommend that you investigate third-party solutions, but only as an addition to your existing suite of regular DBCC checks!

Some third-party products we are aware of include:

- ProActive DBA and the Disaster Recovery Toolset, from [White Sands Technology](#) (of course!)
- SQL-BackTrack and Database Verifier for Sybase, from [BMC Software](#)

## Finding Data Corruption the Hard Way (From Users)

Running regular DBCCs is your first and best line of defense against database corruption. Unfortunately, however, even if you run regular DBCCs, all too often you will find out about database corruption from irate users who start getting application error messages because of the corruption.

In other cases, the corruption might be such that your entire server may be brought down. Obviously, the server is designed to prevent this from happening, but as software is never perfect and cannot account for every possible case, this too can and does happen on very rare occasions in the real world.

It is in these sorts of situations that you most need a disaster recovery plan, since a simpler solution utilizing backup/restore or replication may not be feasible, either because it will require taking the database offline for too much time, and/or because it will not be able to preserve the latest changes made to the database since the last backup.

---

## When !@#&% (Corruption) Happens

Let's say you, being a conscientious DBA, have set up a daily DBCC script to run against each of your production databases. Let's also say you have set up mirrored disk arrays, replication and backups so you are making regular copies of your data at the physical disk, logical database and transaction levels.

So why is your phone ringing off the hook with dozens of angry users complaining about page and buffer errors? Because, my friend, !@#&% (corruption) happens!

We hope this situation never happens to you. But if it does, we want you to have a good plan for dealing with it. In this section, we submit a set of recommended steps for determining the cause of the corruption, and isolating the exact extent of corruption in your server.

## Investigating the Cause of the Corruption

Before even thinking about how to fix corruption, it is essential to find out exactly what caused it. This seemingly obvious step may at times be overlooked in an effort to quickly bring the corrupted database or table back on-line; however, since of course you don't want the corruption to keep occurring, it is best to isolate and fix the cause before proceeding with a fix for the symptom.

In the section starting on page 6, we discussed the various causes of corruption. In general, you will want to follow these steps:

- 1) Check for disk hardware failures, such as an entire disk or array having failed, or a power failure that was not recovered from properly. If you have a mirrored disk subsystem with automatic failover, failure of an entire disk may not even be a critical failure, and you may be able to bring a replacement disk or disks online without users even noticing. On the other hand, if something went wrong during the failover or restoral process, corruption may have worked its way into the primary disk(s), which will now need to be repaired.
- 2) Check for other hardware failures; methods for doing this might include:
  - Checking operating system error logs for any failure notifications;<sup>4</sup>
  - Checking other running applications on the database server's host system for failures;
  - Running hardware diagnostics to ensure the rest of the system is operating properly.
- 3) Check for any operating system failures:
  - Checking operating system error logs for any failure notifications;
  - Checking other running applications on the database server's host system for failures;
- 4) Finally, check the Sybase dataserver's error log for a full description of the failure(s) which it encountered. We provide explanations of some of the more common error messages starting on page 26.

The point is to check the underlying hardware and software which supports the Sybase server first, in case any problems turn out to be what originally caused the corruption to surface.

---

<sup>4</sup> Steps for viewing the OS error logs are described in the topic "[Checking the Operating System Error Log](#)" in Sybase's ASE Troubleshooting and Error Messages Guide.

Additionally, failures can occur in the hardware or in the operating system which go undetected, so you may not be able to find any evidence that this has occurred. In this case, you may have no alternative but to proceed with fixing the corruption and then waiting to see if it happens again.

## **Determining the Extent of the Corruption**

Once you (hopefully) have a handle on what caused your corruption, the next step is to find out just how bad it is. This is best done in three general steps:

- First, determine which database(s) are affected by the corruption;
- Next, identify which table(s) in each database are affected;
- Finally, determine which and how many pages are affected within the corrupt table(s).

By starting at a high level and working down, you can determine whether the corruption is so bad that you will save the most time by just reloading the entire server or an entire database, or if the system can “limp along” while you make smaller repairs to affected areas of the database or individual tables.

To find out the full extent of the corruption, you will want to check each database as thoroughly as possible to determine where corruption exists. To do this, use the same methods as described starting on page 12:

- Run full DBCCs against the affected database and, if possible, against all the other databases;
- Use DBCC PAGE and third-party tools such as White Sands Technology’s Disaster Recovery Toolset to provide additional verification and cross-checking of your DBCC results.

Initially, to save time or reduce downtime, you may want to run the DBCC checks against a replicated server or against a copy of the database(s) restored from backups. This is often done as a first step in order not to have to bring the production system or databases down to run the checks.

This is a useful way to obtain some information about the corruption. BUT—never assume that whatever you find (whether it’s corruption or a lack thereof) on a restored backup or on a replicated database will be exactly the same as what exists on your production database.

When corruption occurs, it may or may not have made its way into your backups. Also, the corruption may have occurred after the most recent backup was made, in which case the corruption will not exist in the backup.

Since a replicated database is a logical copy of the primary (but not necessarily an exact physical copy), running DBCCs against a replicated database will not catch corruption that occurred only in the primary database. Because the replicated database is a complete, separate database system from the primary, it can experience its own corruption that does not exist in the primary database.

Thus, the best strategy, when corruption occurs, to run DBCCs against the database in question, as well as its replicated database, if any.

## Standard Disaster Recovery Strategies: Benefits, Drawbacks and Limitations

There are several standard disaster recovery strategies known to the DBA community, which can be researched using various resources such as DBA reference manuals, the Internet, Sybase Technical Support or consultants.

Much of this can be found in the Sybase ASE Troubleshooting and Error Messages Guide, Chapter 2, in a section titled [“Avoiding Disaster Through Good DBA Practices.”](#)

*Standard recovery techniques may not be enough in an emergency!*

As we shall see in the following sections, while these practices are a good start and form the backbone of a good disaster recovery plan, they are not always a complete solution and may not cover you completely.

### Restart the Database Server

The first thing a DBA might try when faced with corruption is to shut down and restart the database server. This is useful if the corruption existed only in the server's data cache but not on the disk.

We only recommend using this method in cases where you have run full DBCC checks (of course using the "nofix" options as described on page 3) and the checks came up clean with no errors reported.

In such cases, bringing the database server down and back up will typically correct the problem, at least for the moment.

This solution is usually only effective for a certain period of time though, since whatever problem originally caused the corruption is likely to surface again before too long. But it can be a useful way to "buy some time" so you can investigate the problem.

### Copy Out and Reload the Data

A standard solution recommended by some when corruption is discovered, is to copy the data out to another database using SELECT INTO, or to make a copy of the data using BCP; then, drop the table and reload it from the copy that was made.

The problem with doing this is that the corruption may prevent SELECT or BCP from being able to access all the rows in the table—leaving you with a copy of only part of the data.

*SELECT or BCP often does not work on corrupt tables.*

At best, the SELECT or BCP will return an error, or (if a circular loop exists in an object's page chain) it may simply run forever without completing (or at least run until there is no more disk space on the output device and then return an error).

At worst, however, the SELECT or BCP may *appear* to have completed successfully while still not having copied all the rows; this can happen if an object's page chain is incorrectly terminated, causing SELECT or BCP to believe it has read all the pages in the table, when there are really other (stray) pages in the table's page chain that SELECT or BCP cannot see.

So, when corruption exists in your database, it can be very risky to rely on SELECT or BCP to retrieve your data from the corrupt database. If you can verify that the row count is correct in the copy of the table you made, then it is probably a good copy. But, if you are not sure how many rows should be in the table, you are better off not relying on these methods to recover your data.

White Sands's Disaster Recovery Toolset includes powerful export tools that can recover *all* of the pages in a corrupt table, not just the ones accessible to Sybase. This gives you more options when faced with corruption—you don't have to live with being able to recover only part of a corrupt table; instead, you can save out all the rows that can be physically accessed, and then reload the corrupt table, load the data into another table for comparison, etc.

### **Drop and Rebuild the Corrupt Object**

If the corruption exists only on an index, you can often simply drop and recreate the affected index to solve the problem.

But **be careful**—if allocation or page chain errors exist on any tables in the database, dropping or recreating objects can cause further corruption and/or **loss of data!** Before dropping or rebuilding any objects, you should be certain that no allocation errors exist that might cause data in other objects to be affected.

We recommend you follow the procedure on page 4 for making an exact binary copy of the database onto a test server, and attempting the fix there to see whether and how well it works, before making the same fix on your production system.

Rebuilding an index also requires locking the table or making it otherwise unavailable for the duration of the index rebuild, so this may cause significant user delays if the affected table is very large. If you cannot afford the downtime to rebuild the object, you may have to look at other options.

Rebuilding a clustered index on an APL (allpages-locked) table presents another challenge: if you want to use the SORTED\_DATA option to rebuild the index (which saves significant time because it only rebuilds the index layers of the table, without copying and resorting the data layer), you must ensure that the index key values do not change while the index is being rebuilt. This may require keeping users out of the database or putting the database in single-user mode, neither of which option may be very attractive in today's high-availability environments.

### **Switch In Your Mirrored Disk**

On systems with mirrored disks, the mirror provides 100% redundancy against failure of the main disk. This should provide complete protection against erroneous (corrupt) data which may get written to one of the disks—you can move the mirrored disk into the primary position, and then at your leisure (relatively speaking) you can install a new mirrored disk to replace the original.

However, when the corruption occurs at the operating system level, the corrupt data will get written to both the primary and the mirrored disks, rendering them equally useless to your production operation. In this case, you must look for another solution.

### **Restore From a Backup**

Restoring your database from a backup made before the corruption existed is generally the least labor-intensive and most reliable method of removing corruption from a database. However, it has several potential (and likely) drawbacks:

- Current releases of Sybase ASE (through Version 12.5) only support backup and restoral of an entire database, not of specific objects within a database,

so the downtime and/or disk space required to do this may be prohibitive in your situation;

- You may not be able to recover any of the changes made since the last backup without reintroducing the corruption into the system;
- The corruption may have existed and gone undetected for some time, meaning that the last backup made which did not contain the corruption may be too old to be useful.

In short, because of the downtime requirements and because the corruption may have worked its way into the backup stream at some unknown time in the past, restoring from backups often becomes a last resort in dealing with corruption.

### **Restore From a Replicated Database**

Restoring the affected table(s) from a replicated database offers the benefit of being able to restore a much more up-to-date copy of the tables than the most recent backup is likely to contain.

Additionally, you can restore only the affected tables from the replicated server, without having to bring the entire database down to do the procedure. This may be a significant benefit over restoring from a database backup.

If you will be replacing an entire corrupt table with a good copy from a replicated database, you will need sufficient free space to load the new copy of the table into the database (that is, if you want to keep the table on-line while you load the copy in); depending on the size of the table, this might be prohibitive.

### **Use DBCC to Repair the Corruption**

*Be careful when using DBCC so as not to lose data!*

Various DBCC commands have options to let you repair corrupted areas of a table or database when it discovers these problems. Using DBCC has the advantages that it is relatively simple to execute, and it will often return the database and all its tables to a known good state.

However, here are some things to consider when using DBCC with the “fix” option:

- DBCC may drop (delete) a large amount of data from any tables that are affected by corruption, and you will not know beforehand that it is going to do so;
- DBCC may not be able to fix the corruption, and if so, will not do anything to correct it;
- Certain DBCC commands require that you put the database in single-user mode to repair corruption.

For example, when DBCC encounters a broken page chain, its solution may be to sever the page chain at that point. If it encounters any further pages that should be part of the object’s page chain, it typically deallocates all of the pages, thus potentially dropping thousands of rows from the table under the guise of “repairing” the corruption.

If you do decide to use DBCC to fix the corruption, make sure you do the following things:

- Before using DBCC to fix the corruption, make an “exact” binary dump of the database so you can restore it to its current state if the DBCC repairs do not work (or make things worse);
- Before using DBCC, obtain row counts on all the affected tables using as many indexes as possible (because you may get different row counts using different indexes—the largest row count is generally the correct one), so that after DBCC repairs the table(s), you can see whether any rows were dropped.
- Load the “exact” binary dump onto a test server and run DBCCs with the “fix” option there.

---

## Advanced Data Recovery Techniques

When standard disaster recovery methods fail or have limitations which make them unusable in your situation, you must turn to more advanced techniques.

Currently, the only way we know of to go beyond the standard Sybase recovery facilities is by using White Sands Technology’s **Disaster Recovery Toolset**, which provides advanced data recovery and data repair tools for on-line disaster recovery of databases.

The benefits of using the Disaster Recovery Toolset’s advanced recovery and repair features are:

- You can recover all of the most up-to-date data in a table, even from areas that lie beyond the corruption and are inaccessible to Sybase;
- You can target specific areas of the database to repair, without having to reload entire tables or the whole database;
- Repairs can be made to your data while the database and all tables remain completely on-line;
- You can fix *any* type of corruption (except an actual hardware failure), not just the few types correctable by DBCC.

The only real drawback to using these techniques is that you have to be very knowledgeable in Sybase’s data structure<sup>5</sup> to make on-line repairs to a production database—otherwise, you are putting your data at even more risk.

Fortunately, White Sands Technology’s Technical Support staff is a valuable resource with great experience in data recovery. With our help, you will be able to evaluate the state of your database and weigh the risks, if any, in performing the recovery, before you proceed.

### General Corruption Repair Strategies

The general approach to use when attempting to repair a corrupt Sybase table is:

---

<sup>5</sup> See the references starting on page 26 for more information on learning about Sybase database storage structure.

- Repair any affected data-layer pages; in APL (allpages-locked) tables, these exist in a doubly-linked page chain, so the page chain pointers may need to be repaired in addition to any other corrupt information on the affected pages;
- Repair any allocation map, OAM or GAM page inconsistencies, so that the pages belonging to the corrupt object(s) can be “seen” properly by Sybase;
- Rebuild or repair any indexes affected either by the original corruption or by the repairs made to the data layer of the table.

These steps are discussed in more detail in the following sections.

## **Repair the Data-Layer Pages**

The first step in repairing a corrupt table is to repair the data layer of the table; once this has been done, then as a last resort you will at least be able to perform a table-scan query to retrieve all the rows from the table for later reloading into a good (non-corrupt) database.

The method for repairing the table’s data layer depends on which type of table it is: allpages-locked (APL) or data-only locked (DOL), also known as RLL (for row-level locked). The locking strategy used by a table greatly affects the data structure of the table, and has a number of implications for how the table must be repaired.

### **Repairing APL Tables**

For allpages-locked (APL) tables (which are the only kind of tables supported on pre-11.9 versions of Sybase), repairing the data layer entails repairing any breaks in the page chain of the data layer. Allocation map and OAM page inconsistencies generally do not affect the ability of the server to walk a table’s page chain, so you can deal with any allocation errors afterward.

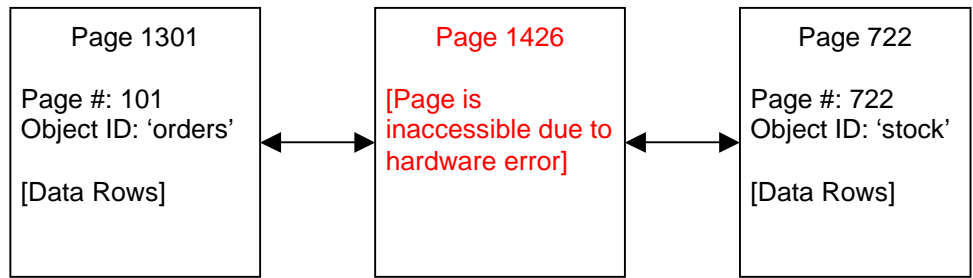
White Sands’s Disaster Recovery Toolset makes this process much easier. Its Page Chain Consistency Checker produces a simple report that shows you where the breaks in a table’s page chain are, and what pages should be linked in where the breaks exist. You can use this information to repair the affected pages, thus rejoining the pages into a single intact chain.

The Database Page Editor, also part of the Disaster Recovery Toolset, provides an easy-to-use graphical interface which lets you change the next-page and previous-page pointers in corrupt pages so as to repair the page chain. Keep in mind that you will need to repair both the next-page and the previous-page pointers on any affected pages. Again, the Page Chain Consistency Checker’s report will identify exactly where you will need to make repairs.

### **Dropping a Page From the Page Chain**

If one or more pages are so corrupt that there is no possibility of recovering data on them (or if, due to a hardware error, the pages cannot be recovered or repaired), you may want to drop the pages completely out of the table’s page chain.

This can be done as shown in the diagram below:



For each corrupt page (such as page 1426 in the above example), you will need to make repairs in two places:

- Change the *next-page* pointer of the page in the page chain before the damaged page, so it points to the page in the chain after the damaged page;
- Change the *previous-page* pointer of the page in the page chain after the damaged page, so it points to the page in the chain before the damaged page.

In the above example, the repairs would be made as follows:



Then, after making the repairs to the page chain, the page that was dropped out of the page chain (page 1426 in this example) would be deallocated by marking it unused in the allocation map page.

Of course, after doing this, you will need to repair or rebuild any affected indexes also.

Before performing the above steps to repair the table, you will want to try to save the row data from the affected page(s).

White Sands's Disaster Recovery Toolset provides a useful facility for doing this: its Page Detail Window lets you build an ISQL script of INSERT statements from any single page or a page chain fragment. Additionally, the Disaster Recovery Toolset can scan the entire database to locate any pages or page chain fragments that are not linked to the table, so you can recover the rows from those pages as well (even if Sybase cannot access the data on those pages).

### Repairing DOL Tables

Repair of the data layer of a data-only locked (DOL) table is quite different from repairing an APL table.

For DOL tables, allocation maps and OAM pages are critical to being able to scan a table; these tables are essentially heaps and have no page chain at the

data level (that is, the page chain pointer values may or may not be valid and in any case are not used by the server).

The access method used by the server to scan a DOL table is called an *OAM scan*, and proceeds like this:

- Look up the table's first OAM page number by retrieving the *doampg* value from the *sysindexes* table for the heap (which is the row where *indid=0*)
- For each allocation unit referenced by the OAM page:
  - Read the allocation map page for that allocation unit;
  - For all data pages belonging to the desired table and marked as being in use, read and process those data pages .
- After processing all allocation units referenced by the OAM page, check the OAM page's next-page pointer to go to the next OAM page, if any, in the OAM chain.

Thus, when corruption occurs on a page, you need only repair that page, or drop it completely from the table; there are no page chain pointers to be concerned with.

To drop a page from the data layer of a DOL table, you need only deallocate the page from its allocation map and update the OAM page to reflect the deallocated page; these steps are described in the next section. Then, you may need to update or rebuild any affected indexes.

### Repairing Allocation Errors

The *allocation map* page in Sybase keeps track of the allocated pages in a 256-page block. Allocation map pages exist every 256 pages in every database—at page 0, page 256, etc. The allocation map has a 48-byte page header, followed by 32 structures of 16 bytes each—one structure for every 8-page extent contained within the allocation unit (32 extents x 8 pages per extent = 256 pages in the allocation unit).

Here is a section from a page dump of an allocation page, as presented by the Disaster Recovery Toolset's Page Detail window:

```
Page 39680 - Page Type: Allocation Page
Page not found in Cache.

-----Page Header Information-----
Page Number: 39680      Next Page: 3      Prev Page: 24510464
Object ID: 99          Index ID: 0      Level: 0
Timestamp: 0x000000000000
Free Offset: 0          Next Row Number: 11      Min Row Len.: 0
Status Bits: [None]

Extents mapped by this allocation page:
  Extent 39,688: APL Table 'dbo.customer' (8 of 8 pages used) NextX=1, PrevX=0
  Extent 39,696: APL Table 'dbo.customer' (8 of 8 pages used) NextX=1, PrevX=0
  Extent 39,704: APL Table 'dbo.customer' (8 of 8 pages used) NextX=1, PrevX=0
  Extent 39,712: APL Table 'dbo.customer' (8 of 8 pages used) NextX=1, PrevX=0
  Extent 39,720: APL Table 'dbo.customer' (8 of 8 pages used) NextX=1, PrevX=0
```

Here is the corresponding section of the hex dump from the page:

0:	00 00 9B 00 00 00 00 03 01 76 00 00 00 00 00 63	.....v.....c
10:	00 00 00 00 00 00 0B 00 00 00 00 00 00 00 00 00	.....
20:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
30:	00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00	.....
40:	00 00 00 01 00 00 00 00 04 C4 C0 F4 FF 00 00 00	.....
50:	00 00 00 01 00 00 00 00 04 C4 C0 F4 FF 00 00 00	.....
60:	00 00 00 01 00 00 00 00 04 C4 C0 F4 FF 00 00 00	.....
70:	00 00 00 01 00 00 00 00 04 C4 C0 F4 FF 00 00 00	.....

The first three 16-byte rows comprise the allocation page header; then, subsequent rows describe the usage of each extent within the allocation unit.

The third extent, for example, at offset 50 on the allocation page, is owned by object 04C4C0F4 hex, or 80003316 decimal, and the FF hex byte in the fourth position from the end of the row is the *page-usage byte*, which in this case indicates that all 8 pages in that extent are in use by the object.

The page-usage byte contains one bit for each page in the extent. The low bit (with a value of 1) corresponds to the first page in the extent, and the high bit (with a value of 128, or 80 hex) represents the last page in the extent.

By modifying the page-usage bits, you can allocate or deallocate pages in an extent.

Keep in mind that if you manually add or remove pages to a table in this manner, you will also need to update the OAM page for the affected table or index. DBCC can usually accomplish this, so that you do not have to manually update the OAM page values.

### Repairing a Corrupt Index

When faced with repairing an index, you should take into account the following:

- What is the extent of the corruption on the index?
- How long would it take to simply drop and recreate the index? (For a clustered index on an APL table, you can use the SORTED\_DATA option to save time, as long as the key-value order of the rows will not change while you are rebuilding the index)
- How will users be affected during the index rebuild (since the table will be locked while it is in progress)?

Examining these issues lets you decide whether it is better to rebuild or reload the entire affected object, or perform repairs on a more narrowly-targeted basis.

The following sections give more detailed information on repairing specific error conditions that can occur in Sybase.

### Recovering Damaged Data

Before attempting to repair a corrupt database, it is prudent to make a backup of at least the affected pages, and preferably the entire affected table(s), if time permits. This ensures that you have some kind of recourse should the attempts at repairing the database fail.

*Recovering data with the Disaster Recovery Toolset.*

The best way we know of to recover all the data in a corrupt table is to use White Sands Technology's Disaster Recovery Toolset. It will build an ISQL script from the rows contained on all the pages belonging to a corrupt object. To do this, it performs the following steps:

- First it scans the page chain of the affected object and generates ISQL INSERT statements for all the rows on the pages in the page chain;
- Then, it scans the entire database for any stray, unlinked pages or page chain fragments that may exist due to breaks in the main page chain.

In this manner, the Disaster Recovery Toolset can quickly and reliably recover all the rows that are present in the data layer of a table, regardless of page chain breaks or other errors it may encounter in the process. You can then reload all or part of the table as necessary.

*Recovering data with  
SELECT.*

An alternate, but potentially much slower, option is to work around the corruption by not using the data-layer page chain. This means issuing SELECT statements to access the table by a nonclustered index; for example:

```
SELECT * INTO new_table FROM corrupt_table (INDEX
nc_good_table)
```

This requires that you have a nonclustered index on the table which was not affected by the corruption. If you have such an index, then this method may be able to access all, or almost all, of the rows in the table.

Another, similar option is to use WHERE clauses to work around the corruption:

```
SELECT * FROM table
WHERE key_val >= 0 AND key_val <= 10000

SELECT * FROM table
WHERE key_val >= 10000 AND key_val <= 20000
```

etc.

Use an ORDER BY or forced-index clause to require the optimizer to use the nonclustered index, if necessary.

*Recovering data by  
modifying sysindexes.*

Yet another method we have heard of is using the DBCC PGLINKAGE<sup>6</sup> command to locate the ends of the page chain fragments in an object having a broken page chain, then hand-modifying the *sysindexes* system table's *first* column so as to make the broken parts of the page chain (temporarily) accessible to Sybase via a clustered index scan.

The downside to using this method is that it requires that you know at least one page number in each fragment of the broken page chain. If you know one page number in a particular page chain fragment, you can use DBCC PGLINKAGE to trace the page chain forward and backward from that page until it finds the break in the page chain.

Unfortunately, if you don't know where the page chain fragments are, it can be difficult to locate them. The Disaster Recovery Toolset makes this job much easier by automatically locating all page chain fragments in a corrupt table for you.

---

<sup>6</sup> The syntax of DBCC PGLINKAGE is "semi-undocumented"; for more information, refer to <http://my.sybase.com/detail?id=20405>.

## Understanding Sybase Error Messages

Most of the error messages you will see in your Sybase server's error log which pertain to data corruption are intimately related to the data storage structure used by Sybase. So, to best understand and diagnose corruption in your database, you will need a thorough knowledge of how Sybase stores data internally.

*Learning about Sybase storage structure.*

Some good descriptions of Sybase's data storage structure, available in print or on the Web, are listed below:

- ProActive DBA User's Manual, Chapter 2 (Overview of SQL Server Data Storage)
- Sybase Adaptive Server Enterprise [Performance & Tuning Guide](#), Chapter 7 (Data Storage) and Chapter 9 (How Indexes Work)
- [Sybase Internals](#) (Kirkwood, John; International Thomson Computer Press, 1996), Chapter 8 (Storage)

### About Sybase Error Logs

*SQL shown in error logs may or may not be relevant.*

Often, when a user receives a corruption-related error, detailed information about the error will be written to the Sybase server's error log file. Here you may see information about the user process that received the error, along with all or part of the SQL that was being executed. Be aware that the SQL statement(s) shown here may or may not be the exact ones that triggered the error. The SQL shown in the error log (often as part of a stack trace) comes from the process's *network receive buffer*, which may only contain the last network packet received and not the entire SQL batch that was being executed.

In short, do not necessarily count on the SQL text shown in the error log as being the exact SQL command that triggered the error, although it may be a good place to start looking.

*Understanding object IDs*

Often, corruption-related error messages report database and object ID numbers instead of names. However, you can easily convert the IDs to the names by using the *db\_name()* and *object\_name()* system functions. Refer to your Sybase Reference Manuals for more information.

Also note that object ID 99 is sometimes seen in error messages; the *object\_name()* function reports its name as 'ALLOCATION'. This is an internal ID (not appearing in *sysobjects*) used for allocation map pages, which exist at every 256<sup>th</sup> page in every database. Any error messages in Sybase which mention object ID 99 are related to accessing allocation map pages in a database.

The following sections describe some of the more common corruption-related errors reported by Sybase database servers.

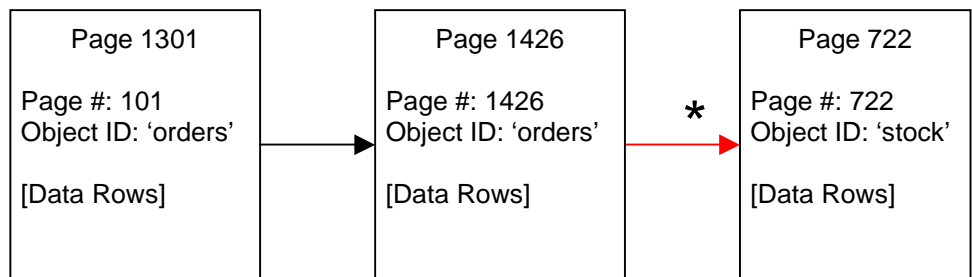
### Error 605

*An attempt was made to fetch logical page '722' in database 'Sales' from cache 'default data cache'. Page belongs to object with id '112003431', not to object 'orders'.*

This is one of the more common corruption-related errors, but the true cause of it can be difficult to discern from the information that is reported in the error log.

The 605 error, categorized under Access Methods Errors, indicates that the server was trying to access a table ('orders' in the above example), but the page

that was read had the object ID of a different object in its page header, as shown in the example below:



\* Corrupt page chain link

By using the *object\_name()* system function, you can find the name of the table whose object ID is given in the error message (112003431 in this example).

#### *Causes of 605 errors.*

In essence, this message means that the page that was read contained the wrong object ID value in its page header. This can have a number of causes:

- The page chain of table 'orders' or one of its indexes is corrupt and points to a page belonging to another object (in other words, page 722 is a valid page for some other table—in this case, the table having object ID 112003431);
- The page being read (page 722 in this example) used to belong to table 'orders', and in fact is still allocated to that object and is part of its page chain, but the page was erroneously overwritten with data from the other object (object ID 112003431);
- The data on the desired page was corrupted in some other fashion (and may still contain some or all of the original data); this is most likely the case if the object ID shown (here, 112003431) is not a valid object in this database.

### **Diagnosing 605 Errors**

To properly diagnose and correct this error without loss of data, the following questions must be answered accurately:

- Does page 722 contain data for table 'orders' or for the other table (the table with object ID 112003431)? To answer this, you will want to do the following:
  - First, look at the page header, to see what other pages, if any, page 722 is linked to. If it is linked to pages from the other table, chances are it does not belong to table 'orders'. If it is linked to pages which also belong to table 'orders', it probably belongs with that table.
  - Next, look at the other fields in page 722's page header, such as the minimum row length and index ID, to see if they match the page chain for table 'orders'—if they do not, the page probably contains data for the other object.
- Do the page chains for any other tables or indexes, or any *sysindexes* rows, point to the affected page (here, page 722)? If so, page 722 may have been allocated to table 'orders' incorrectly (i.e. when that page was already in use

by another object). The other object will need to be identified and repaired as well as the 'orders' table.

To answer these questions, you will first want to run DBCC commands CHECKTABLE and TABLEALLOC against the affected table(s) to verify the integrity of the indexes and other structures of those tables. It is basically a good "first shot" at identifying which tables have corruption.

The Page Detail Window in White Sands's Disaster Recovery Toolset can make answering these questions much easier—it provides a formatted dump of the rows on a given database page. From this, you can identify whether the row data looks valid or not (as determined by the column definition of the table which the page is supposed to belong to—the 'orders' table in this example).

*The Disaster Recovery Toolset helps diagnose broken page chains.*

In addition, the Disaster Recovery Toolset lets you quickly and easily identify which page(s) of which table(s) point to the affected page, so you can narrow down the scope of the corruption. It will also produce a report showing you where any breaks in the object's page chain are, so you can repair the affected areas or work around them in recovering the data from the table.

Finally, the Disaster Recovery Toolset contains tools which let you safely and accurately extract **all** the rows from the affected table, even near the location of the corruption, in case you need to truncate and reload the table.

### **Repairing 605 Errors**

Once you know the extent of the corruption relating to a 605 error, the steps involved in fixing it can be fairly simple—that is, if the damage is localized to one part of the affected table.

Typically, you will need to repair the page that pointed to the page number reported in the 605 error, so that its "next-page" pointer points to a valid page. You will also need to fix up the "previous-page" pointer of the true next page, so the page-chain linkage is intact in both directions.

Then, if any indexes pointed to the corrupt page, these indexes will have to be rebuilt, or the index pages will have to be modified (by having rows deleted or modified) so they are no longer pointing to the corrupt page.

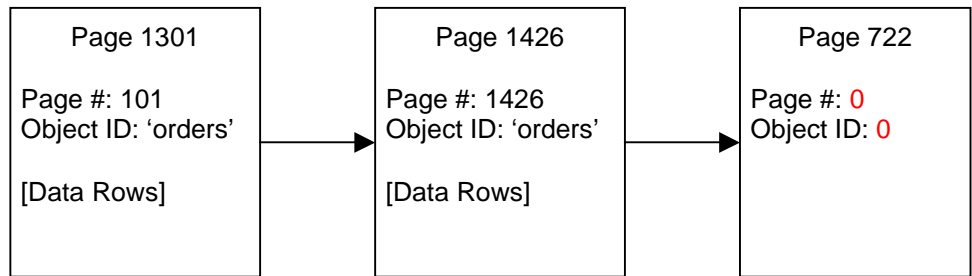
Finally, if any rows were dropped as a result of fixing the corruption, they will need to be reinserted into the table so that all the original data remains present in the affected table(s).

### **Error 692**

*Uninitialized logical page '722' was read while accessing object '112003430' in database '11'. Please contact Sybase Technical Support.*

This error is relatively rare, which is fortunate because it is one of the more serious corruption-related errors (as evidenced by the request to contact Technical Support at the end of the message).

It occurs when a page is read from disk, and the page number and object ID values found in its page header are both zero (thus implying that the rest of the page probably contains zero bytes as well). The following diagram illustrates this condition:



Possible causes of this error include:

- The page in question (and possibly others) was overwritten with zero bytes by the operating system's disk-formatting command or another utility such as 'dd';
- An error was made in the configuration of the disks or disk partitions used by Sybase, causing the data in one or more partitions to inadvertently get formatted or overwritten (systems not utilizing a Logical Volume Manager, or LVM, are particularly prone to this);
- Due to a bug or configuration error, the operating system read some uninitialized (zero) data from the wrong location on the disk, or from the wrong disk altogether.
- Due to a hardware failure, the disk controller or drive returned incorrect (zero-byte) data but failed to signal the failure to the operating system.

### Diagnosing 692 Errors

With 692 errors, depending on what actually caused the error, it is likely that a whole section of a disk (or at worst, an entire disk) is affected, rather than the problem being localized to a single page or several pages.

The first thing you should do is run full DBCCs against the database, and against any other databases which share any of the same disks with the affected database.

The Disaster Recovery Toolset's Database Page Map is also helpful in getting a "bird's-eye" view of the database so you can see what areas are affected by the corruption.

If you discover that entire databases or large areas of databases have been zeroed out or are inaccessible, your only recourse may be to reload the entire database(s) from backups—i.e. "surgical" repair of individual pages of the database may not be possible.

On the other hand, if a small number of pages were corrupted, it is usually possible to recover most of the data, although you may have to go to a backup tape in order to recover the rows on the page(s) that were zeroed out.

### Repairing 692 Errors

Usually when a 692 error occurs, the entire corrupt page's contents are no longer accessible. This may be due to a hardware failure (such as a bad spot on a disk) which should be investigated immediately!

In any case, to repair the error, the corrupt page must be dropped out of its page chain (if any) and deallocated. Then, if possible, the rows that were on that page

should be reloaded from some other source, such as a database backup loaded onto another machine.

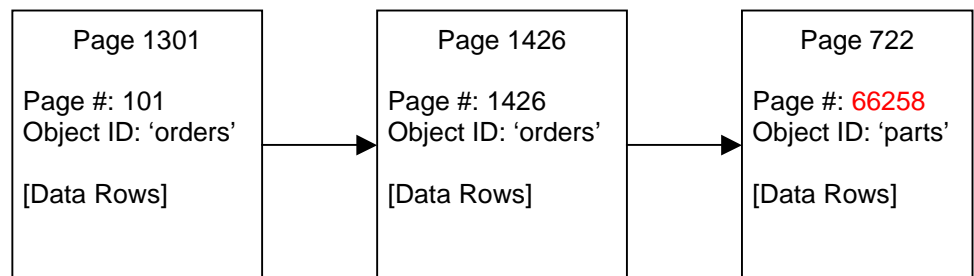
If the corrupt page was a nonclustered index leaf or intermediate page, the page cannot be simply dropped, because the index structure will no longer be consistent. Furthermore, it can be very difficult to reconstruct the index page unless you can load a fairly recent backup copy of the database onto another server—this is because there is usually no other way to know which rows (i.e. which nonclustered index key values) were stored on the page before it became corrupted. For these reasons, rebuilding the nonclustered index is usually your only option when a 692 error has occurred on a nonclustered index page.

If the corrupt page was a clustered index leaf (data) page, it may be easier to determine which rows were on the page, but you may still need to refer to a recent backup copy of the database to obtain the exact contents of the rows.

## Error 695

*An attempt was made to read logical page '722' for object '112003430' in database '11' from disk. Wrong logical page '66258' was brought into cache 'default data cache'.*

This message indicates that the server was trying to access a database page (page number 722 in this example), and because the page was not found in cache, it attempted to read the page from disk. The actual page that was read, however, had the wrong page number (here, 66258) in its page header, as shown in the following diagram:



The database and object IDs for the object the server was trying to access are shown in the error message; using the `db_id()` and `object_id()` T-SQL system functions, you can find the names of the database and table in question.

The cause of this error is usually one or more of the following:

- The page data for a particular database page (page 66258 in this example) was written to the wrong location in the database (here, it was written to the location of page 722) due to an error or bug in Sybase;
- The page data for a database page was written to the wrong location on the device, or to the wrong device altogether, by the operating system;
- The page data on the disk is actually correct, but the server, in attempting to read data from database page 722, read the data from the wrong location and thus received the data for page 66258;
- The data on the disk is correct, but the operating system read the requested page from the wrong location or the wrong device;

- The data on page 722 was corrupted somehow, and may still contain some or all of the original data.

### Diagnosing 695 Errors

This error can be a little more difficult than others to track down and correct, because it may involve failures in the LVM (logical volume manager), in the operating system or in the hardware. If the Sybase database server cannot reliably read or write to the correct location on the disk, any attempts at repairing the corruption at that location may not work either.

If, however, it is just a case of the wrong data having been written to that page, the error can probably be repaired. You must gather information pertaining to two pages in the database:

- The page whose number is given by the location at which the page was read (page 722 in the above example), which we refer to as the “*target page*”;
- The page whose number was found in the page header (page 66258 in this example); we’ll call this the “*source page*.”

The data read by the server for the target page may belong to one or the other of these pages, and to one or other of the objects which own each of these pages. To find out, you must gather the following information:

- Using, for example, DBCC PAGE or the Disaster Recovery Toolset’s Page Detail Window, examine the page to determine whether the data on the page really belongs to the source page or the target page. This can be determined by looking at the “minimum row length,” “index ID” and other fields in the page header, and also by examining the contents of the rows on the page.
- Also examine the contents of the real source page (page 66258 in the above example) to determine whether it can be read correctly by the server and whether the table that owns it is also corrupt.
- Finally, if the source and target pages are owned by two different tables, perform full DBCCs and use the Disaster Recovery Toolset’s Page Chain Consistency Checker to identify any other breaks in the objects’ page chains or any other corruption in either of the two tables.

The above steps should tell you the extent of the corruption and what object the damaged page really belongs to.

### Repairing 695 Errors

Usually with 695 errors, one page (the target page) has been overwritten with the contents of another page (the source page). This has two implications:

- The original contents of the target page may have been overwritten and are lost (except as may exist in any backups that did not contain the corruption);
- If the current contents of the source and target pages differ, you will need to decide which contains the most current version of the data that you will wish to keep.

Once you know what to do with the data on the corrupt page, the steps to take will include one or more of the following:

- If the corrupt page is a duplicate (i.e. you don't need to preserve the rows on it), deallocate the page by removing it from its allocation page and possibly the OAM and GAM pages;
- If you will be keeping the corrupt page, repair it by changing the page number in its page header;
- Remove the corrupt page from one or both affected objects' page chains, or repair the page chains as necessary so the corrupt page is included in only the correct object's page chain;
- If you can identify which rows were on the lost page, and if you have a recent enough database backup, you can reload the lost rows by loading the backup onto a test server, for example, selecting the rows in question, and reinserting the rows back into the production database;
- Repair or rebuild any affected indexes.

## **Error 697**

*An attempt was made to fetch logical page '722' for object '112003430' in database '11' from cache 'default data cache'. Wrong logical page '66258' was found in cache.*

This message is very similar to error 695—it indicates that the server tried to access a database page, and received the wrong page number.

The difference here is that the page was already found in cache, but had the wrong page number in its page header. If a 694, 695 or other error occurred previously on the same page, then further attempts to access the page will result in 697 errors, so the 697 error is really just a “reminder” of the corruption that was found on the disk.

The rest of this write-up assumes the 697 error occurred on a page without previous 695 or other errors; in this case, it indicates the corruption probably occurred in memory alone, which is indicative of either a hardware (RAM) problem that went undetected when it first occurred, or a serious Sybase bug.

## **Diagnosing 697 Errors**

Recycling (rebooting) the Sybase server is probably your best first strategy for correcting 697 errors. Due to their nature, 697 errors should not persist across a server reboot. If after rebooting the problem does not recur, you can probably continue using the server for some time—with the following caveats:

- You will want to run full hardware diagnostics on the server machine, to make sure there have been no RAM or other component failures;
- Make sure you are using the correct (usually most recent) releases of the Sybase and operating system software, since a problem like this will usually not go for very long without being uncovered and corrected by the software manufacturer;
- You may want to formulate a plan to migrate production work off of that server, if necessary, until you can determine what was causing the problem, because other corruption could occur at any time, and could go undetected for some time as well.

## Repairing 697 Errors

Because 697 errors arise from in-memory corruption rather than corruption on the disk, recycling the server is usually your only remedy. If the 697 errors persist through a reboot, there may be some other problem (bug) in the reporting of this error or related errors.

In this situation, the remedies available for correcting 695 errors (see the previous section) could possibly apply here also, but your best bet would be to check with Sybase Technical Support to be sure.

## Error 806

*Could not find virtual page for logical page 16843724 in database 'Sales'.*

This cryptic little error implies a page could not be found in the server. But what is it really saying?

In actuality, what happened was that somehow the server ended up looking for an out-of-range page number within a database.

The error message is in effect saying that Sybase, for some reason, had a logical page number within a database that it was trying to read, and in doing so it found that there was no corresponding device-relative page number for that logical page; specifically, the logical page number could not be found within the range of any of the device/database fragments listed in the *master..sysusages* system table.

This can have several possible causes:

- The page chain of a particular page in the database was linked to an incorrect, and out-of-range, page number;
- A clustered or nonclustered index row had a pointer to an incorrect, out-of-range page number;
- An incorrect value in *sysindexes* pointed to a root, first or distribution page number that was out of range;
- A more rare case is that one or more *sysusages* entries for the database became corrupt, and while the page number shown in the 806 error message is actually correct, the server thinks it is an out-of-range page number.

## Diagnosing 806 Errors

With an 806 error, the first thing to check is whether the page number shown in the error message is actually a valid page number within the database. The lowest valid logical page number in any database is Page 0 (i.e. no negative page numbers are allowed). The highest valid page number can be found using the following formula:

$$\text{Highest Page \#} = (\text{Database Size in MB} * 512) - 1$$

For example, a 500MB database would have a maximum page number of  $(500 * 512) - 1$ , or 255999; valid page numbers would be anything from 0 to 255999 (but any page numbers 256000 or higher are invalid).

Note that, with 806 errors, you should **not** rely on querying *sysusages* to determine the database size. If you do not know it already, try to verify the

database size from another source besides *sysusages*, since in rare cases it may be corrupt and the size reported by querying it can be misleading.

If the page number in the error message is valid, then your *master..sysusages* table is probably corrupt and will need to be hand-modified using an ISQL tool or other means to make it correct again. Or, if other databases' entries in *sysusages* are affected as well, you may have to reload your *master* database.

If the page number in the 806 error message is not valid, then the problem is probably more localized (but may exist in more than one place). In this case, since the page number in the error is not valid, it is not the true source of the corruption, and you will need to do a little digging to find out exactly where the corruption is.

### Repairing 806 Errors

Unfortunately, the 806 error message itself does not show which table the server was accessing which led to the invalid page number. The server error log may show part or all of the SQL which accessed the invalid page, however, so this may be a starting point. (Note, however, that as mentioned on page 26, the SQL shown in the error message comes from the network receive buffer, and may not reflect the actual SQL command that caused the error.)

Otherwise, you can use DBCC and/or White Sands's Page Chain Consistency Checker to identify which table(s) are corrupt, and which page pointed to the out-of-range page number. The pointer to the invalid page number will probably come from one of two places:

- A break in the page chain of at least one object in the database;
- An entry in *sysindexes* points to the out-of-range page as a first, root or (on pre-11.5 servers) an index distribution page for some object.

Once you know where the bad page pointer(s) are, you can correct them by:

- Hand-modifying the *sysindexes* entries using an ISQL tool; and/or
- Repairing the damaged page chain(s) as described in the section on 605 errors starting on page 26.

### Error 1133

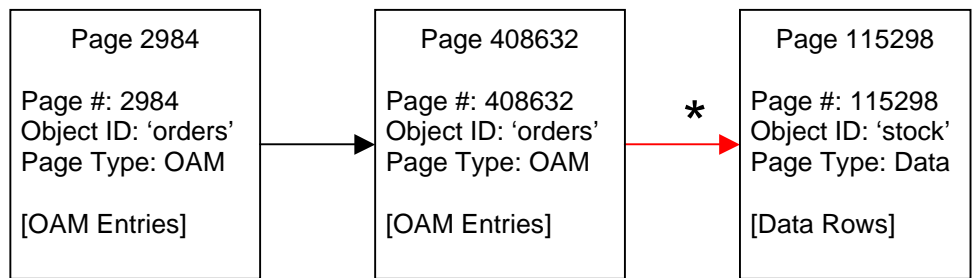
*Page 115298 was expected to be an OAM page for 112003430 and it is not.*

This error occurs when the Sybase server has read what is supposed to be an OAM page for a table or index (specifically, for the table whose object ID is given in the error message), but the page that was read is not actually an OAM page.

The server locates OAM pages in one of two ways:

- The page number of the first OAM page for an object is given by the *ioampg* field (for indexes) or the *doampg* field (for a table's data layer, or heap) in the *sysindexes* system table;
- Subsequent OAM pages are linked from the first OAM page in a circular page chain.

The diagram below gives an example of this:



\* Corrupt page chain link

When the server locates and reads an OAM page using one of the above methods, it then checks the page header of the page that was read in; the “status” field of the page header must contain a certain setting (called PG\_OAMPG), which indicates that the page is in fact an OAM page. If the page does not contain this setting, then an 1133 error is raised.

### Diagnosing 1133 Errors

These errors arise due to one of the following causes:

- An incorrect page pointer exists in *sysindexes* or in the next-page pointer of another OAM page;
- The OAM page used to be at the location that was read, but was overwritten with the contents of another page or with corrupt page data.

To find out which of these is the case, you will need to use DBCC PAGE or White Sands’s Page Detail Window to look at the contents of the page in question (page 115298 in the above example).

The page is likely to belong to some other object, and is also likely not to be an OAM page. Or, the page may be empty. If the page belongs to another object, you will want to run full DBCCs against that object, as it may contain corruption.

You will also need to determine whether the corrupt object listed in the 1133 error message (object ID 112003430 in this example) has a valid first OAM page and (optionally) other OAM pages in a valid page chain. Here again, using DBCC PAGE or White Sands’s Page Detail Window, you can follow the OAM page chain until you reach the first OAM page, and in doing so you can verify the integrity of the next- and previous-page pointers on each OAM page.

Note that if the object only has one OAM page (which it typically does if it is less than about 130MB in size), that OAM page will be linked forward and backward to itself.

### Repairing 1133 Errors

Depending on whether the object’s OAM page chain is intact, you may need to add or remove OAM pages to/from the OAM page chain, and then run DBCC to fix up the allocation errors in the OAM page(s).

---

## Conclusion

Hopefully you now have a better understanding of what causes database corruption in Sybase and other database servers, and the various options you have at your disposal for dealing with it if and when it occurs.

As we have seen, with the right tools and the right information, you are no longer at the mercy of data corruption!

For further information on the White Sands Technology, Inc. products mentioned in this white paper, including ProActive DBA and the Disaster Recovery Toolset, contact us at:

Voice: 1-818-702-9200

Fax: 1-818-702-9100

Web: [www.whitesands.com](http://www.whitesands.com)

Email: [sales@whitesands.com](mailto:sales@whitesands.com)

Finally, we at White Sands would like to thank you for reading this white paper, and we hope you find it beneficial in dealing with data corruption.

If you have any feedback regarding this white paper, feel free to contact us using any of the above methods.

---

# Index

- 1133 errors, 34
- 605 errors, 26
- 692 errors, 28
- 695 errors, 29
- 697 errors, 32
- 806 errors, 32
- 99 (as object ID), 26
- Access Methods errors, 26
- Allocation map pages, 20, 21, 26
- Allpages locking. See APL
- APL tables, 21, 24
- BCP
  - limitations of, 17
- BMC Software, 14
- Bulk copy. See BCP
- Cache
  - corruption in, 32
- Corruption. See Data corruption
- Data corruption
  - causes of, 7
  - definition, 6
  - detecting, 12
  - investigating cause of, 14
  - rebuilding corrupt objects, 18
  - recovering inaccessible data, 24
  - reloading corrupt tables, 17
  - repair strategies, 20
  - transient vs. persistent, 6
  - triage procedure, 2
  - understanding, 6
- data layer, 20
- Database
  - backup and restore, 4, 18
  - making a binary copy of, 4
  - triage procedure, 2
- Database consistency checker. See DBCC
- Database corruption. See Data corruption
- Database Verifier, 14
- Data-only locking. See DOL
- DBCC, 12, 19, 27
  - dangers of, 3
  - list of safe options, 3
  - using to repair corruption, 19
- DBCC PGLINKAGE, 25
- Disaster Recovery Toolset, 14, 20, 21, 35
  - examining pages with, 27
- Disk
  - error correction, 9
  - failures, 8
  - mean time between failures (MTBF), 8
  - mirroring, 10, 18
  - service life, 8

- DOL tables, 22
- Error logs (Sybase), 26
- Error messages (Sybase), 25
- GAM pages, 20, 31
- Hard disk. See Disk
- Hardware failures, 7
- Indexes
  - clustered, 24
  - rebuilding, 5, 18
  - repairing corruption in, 24
- Log. See Transaction log
- Logical volume manager, 28, 30
- Nonclustered index
  - and data recovery, 25
  - repairing, 24
- OAM pages, 20, 21, 31, 34
- Object IDs, 26
- Operating system
  - error log, 15
  - failures in, 10
- Page
  - uninitialized, 28
- Page chain
  - circular loop in, 17
  - corruption in, 27, 33
  - dropping pages from, 21
  - incorrectly terminated, 17
  - repairing, 20, 21
- Power failures, 8
- ProActive DBA, 14, 20, 25, 35
- Receive buffer, 26
- Recovering inaccessible data, 24
- Recovery planning, 11
- Repair strategies, 20
- Replication, 10
  - and checking for corruption, 16
  - restoring from, 19
  - warm standby systems, 13
- SELECT
  - limitations of, 17
  - using to recover data, 24
- SORTED\_DATA option, 24
- sp\_dumpoptimize, 4
- SQL receive buffer, 26
- SQL-Backtrack, 14
- Sybase
  - dataserver error messages, 25
  - error logs, 26
- sysindexes system table, 33
  - corruption in, 34
  - hand-modifying, 25
  - repairing, 34
- sysusages system table, 33
  - corruption in, 33
- Tables
  - rebuilding, 5
- Third-party solutions, 13
- Transaction log, 8, 9
  - backing up, 10

Triage for databases, 2

Truncate log on checkpoint option, 10

Uninterruptible power supply, 8

UPS. See Uninterruptible power supply

White Sands Technology, Inc.

contacting, 35

data recovery tools, 21, 22

products, 14, 20

professional services, 20