Data Corruption and Recovery Issues

CHAPTER 2

IN THIS CHAPTER

By Bob Ward

All the troubleshooting concepts discussed in this chapter relate to one common goal: protection and recovery of your most important asset—*your data*!

What is the key to recovering your data? The answer is simple. Restore from a valid backup. But you bought this book to find out "tips and tricks" for data recovery, and all I have told you is to restore from backup. Is that all? Well, not just that, but I am telling you that restoring from your backup is absolutely the most reliable and consistent method to recover your data. Why? `BACKUP/RESTORE` is SQL Server's primary mechanism for recovering your data in the most reliable and consistent fashion.

I have supported customers for every SQL Server version Microsoft has shipped over a 13-year period and have seen many customers contact technical support without the ability to restore a valid backup. SQL Server has such excellent tools to back up your data (and many ISVs have built products using our VDI API) that there really is no reason not to have a valid backup. It simply takes a well-thought-out strategy and the right hardware to ensure you have a backup to meet your recovery needs. The point I make here is that I will present many advanced features and techniques to recover your data; but in some cases, these would not be needed if you have the proper backups. But what if, despite your best efforts, you cannot restore from a backup? You've come to the right place. Although in some situations I discuss restoring from a backup as the best (and perhaps only) solution to a problem, I present other options specifically designed into SQL Server 2005.

Now let's take a look at how this chapter is organized so you can decide how best to read through this material. I have organized the chapter into three main sections that discuss building your knowledge, data recovery troubleshooting scenarios, and exercises.

If you want to *build your knowledge* in the area of data recovery, focus on this first section. I fill in some gaps from the product documentation on specific *storage internal topics* such as new allocation structure terminology. Second, I make sure you are educated on important *SQL Server 2005 enhancements* in the areas of backup/restore and `DBCC CHECKDB`. Third, I provide some tips and suggestions for *best practices* to avoid data recovery problems focusing on backup/restore, `DBCC CHECKDB`, and the system that supports SQL Server, the operating system, and the hardware. I said I wouldn't talk much about disaster recovery strategies, but I can't help it. One of my jobs at Microsoft is to educate and think of ways for customers to avoid calling technical support, which means thinking of ways to prevent problems. So I spend some time on best practices so that you can avoid using advanced techniques to recover your data.

The next section is all about *troubleshooting*, the main reason you purchased this book. Troubleshooting is all about solving problems. Problems can usually be categorized into various *scenarios*. Therefore, this section is organized into various scenarios that you might encounter, including failures to access, backup, restore, or check consistency on your data. If you want to learn about how to solve problems for specific scenarios, you should read this section. But as with all good books, many types of great technical tips and internal information are woven into this chapter. To teach you about troubleshooting data recovery, I first go over scenarios that require you to recover system databases. I then review how to troubleshoot situations when your user database is inaccessible (for example, your user database is marked `SUSPECT`). As mentioned previously, `BACKUP/RESTORE` is critical to data recovery. But what if it fails? Well, we talk about how to handle some of these situations. The last two subsections focus on database consistency. First, I review certain types of database consistency runtime errors. These are errors that can occur during execution of the most basic T-SQL queries (such as `SELECT`, `INSERT`, `UPDATE`, or `DELETE`) after the database has been successfully opened. Some of these scenarios may require you to use `DBCC CHECKDB`. So, I teach you what to do when `DBCC CHECKDB` reports errors. This is one of the most important tools in your *data recovery toolkit*, so it is important to understand more about how it works, proper usage, and what to do when it reports errors.

Fundamentals

Anyone who is good at solving problems will tell you it is important to know something about "how things work." Therefore, in this section, I help build your knowledge of important internal information for SQL Server 2005 related to the general topic of data recovery. But first, I recommend you review the following sections of SQL Server Books Online. Fundamental information in these sections provides a foundation for what I discuss:

Database Engine Manageability Enhancements

Database Engine Availability Enhancements

Understanding Databases

Database Snapshots

Partitioned Tables and Indexes

Understanding and Managing Transaction Logs

Backing Up and Restoring Databases

Using a Dedicated Administrator Connection

Physical Database Architecture

`ALTER DATABASE` (Transact-SQL)

`BACKUP` (Transact-SQL)

`DBCC CHECKDB` (Transact-SQL)

System Views (Transact-SQL)

SQL Server 2005 Storage Internals

Database and File States

In SQL Server 2000, the state (or status) of the database was at best hard to understand. You had to decode special bits in the `sysdatabases` table, and the behavior of when statuses changed was not consistent. SQL Server 2005 does a nice job of cleaning this up. First, a descriptive state of the database can be found in the `sys.databases` catalog view (in a column called `state`). Second, the meaning and behavior of database states are easy to understand and consistent. For example, when you force a database offline with `ALTER DATABASE SET OFFLINE`, the database goes to the `OFFLINE` state. This was not always the case in SQL 2000.

To understand what these database states mean and how a database can move into each state, look at Figure 2-1.

This does not represent every possible scenario for state change for a database, but it covers the most common scenarios. A database is typically started during server startup, when it is first created, or when it is attached. When this occurs, the state of the database is temporarily changed to `RECOVERING`. If there is a resource problem with the database (such as a failure to open the database files), the state is changed to

`RECOVERY_PENDING`. This state is persisted, so after the resource problem has been corrected, the user must use `ALTER DATABASE SET ONLINE` to start it again.

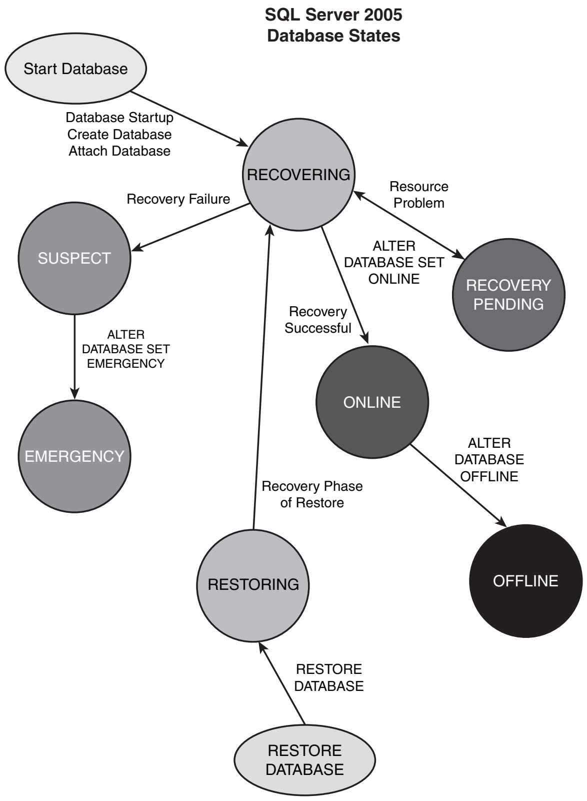


Figure 2-1
Status of the database

If no resource issues exist, the engine tries to run recovery on the database. If recovery is successful, the database state changes to `ONLINE` and is persisted. If recovery fails, the database state is changed to `SUSPECT` but is not persisted. This is a change from previous versions when the `SUSPECT` state was persisted. This caused customers and PSS headaches, because if the server was restarted, the `ERRORLOG` did not contain the original reason for the recovery failure. Now `SUSPECT` is temporary, so if the server is restarted, recovery runs again at database startup. Therefore, the cause for a recovery failure can be investigated from the `ERRORLOG`.

When the database is `ONLINE`, you can take if offline (which closes the database and the database file handles along with it) by running `ALTER DATABASE <dbname> SET OFFLINE`. This syntax changes the state to `OFFLINE`, which is persisted. At any persisted state, you can use `ALTER DATABASE <dbname> SET EMERGENCY` to change the database state to `EMERGENCY`, which is then persisted. This is another nice change from SQL Server 2000, where changing to `EMERGENCY` state required an update to the `sysdatabases` table setting specific bits. This was error-prone, so putting this syntax into `ALTER DATABASE` made perfect sense. As the name implies, setting the database state to `EMERGENCY` makes sense only when you cannot access the database, such as when it is marked `SUSPECT`. We discuss later in the chapter why it sometimes makes sense to use `EMERGENCY`.

It is also important to mention the states that occur when you restore a database. When you restore a database and the database is created as part of loading it from the backup, the database state starts out as `RESTORING`. When all the database pages have been restored, the state is changed to `RECOVERING` as the database is recovered. Then the process is the same as normal database startup.

Database files also have states.

Resource Database

A major change for SQL Server 2005 is the `resource` database. In previous versions of SQL Server 2000, all *executable* system objects were stored in the master database. By executable, I mean any object not used to store data such as stored procedures, views, functions, and triggers. If you look at a typical SQL Server 2000 master database, you will find approximately 1,200 objects. With SQL Server 2005, there are now only about 70. So where are these objects now stored? In a database called the `resource` database. The actual name of this database is `mssqlsystemresource`, but there is a twist. You cannot directly access this database. (In other words, you won't see this in Management Studio, and you get an error if you try to execute `'use mssqlsystemresource'`) So how do you get a list of objects in this database and execute them?

SQL Server 2005 introduces a new ownership concept called schemas. One special reserved schema is called `sys`. By using the `sys` schema, you can reference any system object that is physically stored in the `resource` database in the context of your database. For example, we discuss in the next section a view called `sys.objects`. The view definition for objects is stored in the `resource` database, but you can reference it in a `select` statement in your database like this: `'select * from sys.objects'`. So when you use the `sys` schema name for an object, the engine looks for the definition of the object in the `resource` database.

Why was this change made in SQL Server 2005? There are many reasons, but here is my perspective.

System executable objects are like code from Microsoft, so they are now stored in a protected, read-only database much in a manner similar to a dynamic link library (DLL). The files for this database are not actual DLL executable code. They are two files, `mssqlsystemresource.mdf` and `mssqlsystemresource.ldf`, installed by default in the same directory as the master database files.

These system objects are now in a self-contained location. It is now simple to identify all of them and simpler to replace them. Changes to these objects come in the form of a new database and log file. Copy in a new version of these files for upgrades to these objects. I address versioning and servicing of this database later when I talk about system database recovery.

Referencing a system object is now done in a consistent and clear approach. No more special rules such as "If the name starts with sp_, first check the master database." You just use the sys schema name to reference a system executable object.

Catalog Views and Base System Tables

Around the same time the resource database was created, the system tables were also redesigned. SQL Server has always been a design where the storage of metadata and the interfaces to view them are the same. SQL Server 7.0 did implement INFORMATION_SCHEMA views for ANSI compliance, but most users directly accessed system tables such as sysobjects and sysindexes. But the access is wide open for administrators with the right access. You can query, insert, delete, or update these tables directly. Great for enthusiasts and certain PSS engineers trying to solve some advanced problem, but a nightmare for those trying to ensure consistency of the database. I've seen the root cause of some fairly complex customer case be the result of a mistake made by a customer trying to touch an obscure system table column value.

The sheer scope and number of new features for SQL Server 2005 would require major modifications to a 10-year-old system table architecture. An entirely new system table design was needed to support new features (such as partitions). At the same time, the design of these tables and the interfaces used to access and modify data within them needed to be decoupled. The challenge was to prevent users from "shooting themselves in the foot" while providing a comprehensive metadata interface so that users would have access to everything they need. The result was catalog views. The catalog of the database is all the metadata storage within any database. Approximately 200 views were created and stored in the resource database for users to query the database catalog. Furthermore, support for some operations that formerly required direct system table modification was added to the Transact-SQL language. For example, to put the database in emergency mode, you now use ALTER DATABASE instead of directly changing a system table.

This last point is important, because it highlights a restriction that was put in place as part of this change for catalog views and system tables. Users cannot directly read or change system tables. Your first reaction may be anger or frustration. Why is Microsoft taking away something from me? The intention is not to take anything away from you. The intention is to help you keep a consistent database, provide you with the best interface to view the catalog, and to help avoid backward-compatibility problems should system tables change in future releases. In fact, to help with backward compatibility with SQL Server 2000 users, the "old" system tables such as sysobjects exist in SQL Server 2005 as catalog views.

To help put this together, let's take a look at a few queries using catalog views to get metadata information about the database.

If I want to find out all objects physically stored in my database that I have access to, I run the following query:

```
select * from sys.objects
```

If you were a system administrator and you ran this query in the old pubs sample database (installed on SQL Server 2005), you would get the following results:

```
name
object_id  principal_id schema_id  parent_object_id type type_desc
create_date        modify_date        is_ms_shipped is_published is_schema_
published
--------------------------------------------------------------------------------
------------------------------------------- ---------- ----------- ---------- --
------------- ---- ------------------------------------------------------------
---------------------- ---------------------- ------------ ----------- ---------
----------
CK_ _publisher_ _pub_i_ _023D5A04
37575172  NULL     1       5575058      C  CHECK_CONSTRAINT
2006-02-22 07:01:47.827 2006-02-22 07:01:47.827 0       0       0
CK_ _jobs_ _min_lvl_ _1920BF5C
421576540  NULL     1       373576369    C  CHECK_CONSTRAINT
2006-02-22 07:01:47.873 2006-02-22 07:01:47.873 0       0       0
CK_ _jobs_ _max_lvl_ _1A14E395
437576597  NULL     1       373576369    C  CHECK_CONSTRAINT
2006-02-22 07:01:47.873 2006-02-22 07:01:47.873 0       0       0
CK_emp_id
533576939  NULL     1       501576825    C  CHECK_CONSTRAINT
2006-02-22 07:01:47.873 2006-02-22 07:01:47.873 0       0       0
CK_ _authors_ _au_id_ _7D78A4E7
2105058535 NULL     1       2073058421    C  CHECK_CONSTRAINT
2006-02-22 07:01:47.780 2006-02-22 07:01:47.780 0       0       0
CK_ _authors_ _zip_ _7F60ED59
2137058649 NULL     1       2073058421    C  CHECK_CONSTRAINT
2006-02-22 07:01:47.780 2006-02-22 07:01:47.780 0       0       0
DF_ _authors_ _phone_ _7E6CC920
2121058592 NULL     1       2073058421    D  DEFAULT_CONSTRAINT
2006-02-22 07:01:47.780 2006-02-22 07:01:47.780 0       0       0
DF_ _employee_ _hire_d_ _25869641
629577281  NULL     1       501576825    D  DEFAULT_CONSTRAINT
2006-02-22 07:01:47.890 2006-02-22 07:01:47.890 0       0       0
DF_ _employee_ _job_id_ _20C1E124
549576996  NULL     1       501576825    D  DEFAULT_CONSTRAINT
2006-02-22 07:01:47.873 2006-02-22 07:01:47.873 0       0       0
DF_ _employee_ _job_lv_ _22AA2996
581577110  NULL     1       501576825    D  DEFAULT_CONSTRAINT
2006-02-22 07:01:47.890 2006-02-22 07:01:47.890 0       0       0
DF_ _employee_ _pub_id_ _239E4DCF
597577167  NULL     1       501576825    D  DEFAULT_CONSTRAINT
2006-02-22 07:01:47.890 2006-02-22 07:01:47.890 0       0       0
DF_ _jobs_ _job_desc_ _182C9B23
405576483  NULL     1       373576369    D  DEFAULT_CONSTRAINT
2006-02-22 07:01:47.873 2006-02-22 07:01:47.873 0       0       0
DF_ _publisher_ _count_ _03317E3D
53575229  NULL     1       5575058      D  DEFAULT_CONSTRAINT
2006-02-22 07:01:47.827 2006-02-22 07:01:47.827 0       0       0
DF_ _titles_ _type_ _060DEAE8
101575400  NULL     1       69575286     D  DEFAULT_CONSTRAINT
2006-02-22 07:01:47.840 2006-02-22 07:01:47.840 0       0       0
DF_ _titles_ _pubdate_ _07F6335A
133575514  NULL     1       69575286     D  DEFAULT_CONSTRAINT
2006-02-22 07:01:47.840 2006-02-22 07:01:47.840 0       0       0
```

```
FK_ _titles_ _pub_id_ _07020F21
117575457  NULL     1      69575286     F  FOREIGN_KEY_CONSTRAINT
2006-02-22 07:01:47.840 2006-02-22 07:01:47.840 0       0      0
FK_ _titleauth_ _au_id_ _0AD2A005
181575685  NULL     1      149575571    F  FOREIGN_KEY_CONSTRAINT
2006-02-22 07:01:47.840 2006-02-22 07:01:47.840 0       0      0
FK_ _titleauth_ _title_ _0BC6C43E
197575742  NULL     1      149575571    F  FOREIGN_KEY_CONSTRAINT
2006-02-22 07:01:47.840 2006-02-22 07:01:47.840 0       0      0
FK_ _sales_ _stor_id_ _108B795B
277576027  NULL     1      245575913    F  FOREIGN_KEY_CONSTRAINT
2006-02-22 07:01:47.857 2006-02-22 07:01:47.857 0       0      0
FK_ _sales_ _title_id_ _117F9D94
293576084  NULL     1      245575913    F  FOREIGN_KEY_CONSTRAINT
2006-02-22 07:01:47.857 2006-02-22 07:01:47.857 0       0      0
FK_ _roysched_ _title_ __1367E606
325576198  NULL     1      309576141    F  FOREIGN_KEY_CONSTRAINT
2006-02-22 07:01:47.857 2006-02-22 07:01:47.857 0       0      0
FK_ _discounts_ _stor_ __15502E78
357576312  NULL     1      341576255    F  FOREIGN_KEY_CONSTRAINT
2006-02-22 07:01:47.857 2006-02-22 07:01:47.857 0       0      0
FK_ _employee_ _pub_id_ _24927208
613577224  NULL     1      501576825    F  FOREIGN_KEY_CONSTRAINT
2006-02-22 07:01:47.890 2006-02-22 07:01:47.890 0       0      0
FK_ _employee_ _job_id_ _21B6055D
565577053  NULL     1      501576825    F  FOREIGN_KEY_CONSTRAINT
2006-02-22 07:01:47.890 2006-02-22 07:01:47.890 0       0      0
FK_ _pub_info_ _pub_id_ _1CF15040
485576768  NULL     1      453576654    F  FOREIGN_KEY_CONSTRAINT
2006-02-22 07:01:47.873 2006-02-22 07:01:47.873 0       0      0
queue_messages_1977058079
1993058136 NULL     4      1977058079    IT  INTERNAL_TABLE
2005-10-14 01:36:25.360 2005-10-14 01:36:25.380 1       0      0
queue_messages_2009058193
2025058250 NULL     4      2009058193    IT  INTERNAL_TABLE
2005-10-14 01:36:25.377 2005-10-14 01:36:25.383 1       0      0
queue_messages_2041058307
2057058364 NULL     4      2041058307    IT  INTERNAL_TABLE
2005-10-14 01:36:25.377 2005-10-14 01:36:25.383 1       0      0
byroyalty
677577452  NULL     1      0         P  SQL_STORED_PROCEDURE
2006-02-22 07:01:50.450 2006-02-22 07:01:50.450 0       0      0
reptq1
693577509  NULL     1      0         P  SQL_STORED_PROCEDURE
2006-02-22 07:01:50.483 2006-02-22 07:01:50.483 0       0      0
reptq2
709577566  NULL     1      0         P  SQL_STORED_PROCEDURE
2006-02-22 07:01:50.483 2006-02-22 07:01:50.483 0       0      0
reptq3
725577623  NULL     1      0         P  SQL_STORED_PROCEDURE
2006-02-22 07:01:50.483 2006-02-22 07:01:50.483 0       0      0
UPKCL_auidind
2089058478 NULL     1      2073058421    PK  PRIMARY_KEY_CONSTRAINT
2006-02-22 07:01:47.780 2006-02-22 07:01:47.780 0       0      0
```

```
UPKCL_pubinfo
469576711  NULL     1      453576654    PK  PRIMARY_KEY_CONSTRAINT
2006-02-22 07:01:47.873 2006-02-22 07:01:47.873 0       0      0
PK_emp_id
517576882  NULL     1      501576825    PK  PRIMARY_KEY_CONSTRAINT
2006-02-22 07:01:47.873 2006-02-22 07:01:47.873 0       0      0
PK_ _jobs_ _173876EA
389576426  NULL     1      373576369    PK  PRIMARY_KEY_CONSTRAINT
2006-02-22 07:01:47.857 2006-02-22 07:01:47.857 0       0      0
UPKCL_taind
165575628  NULL     1      149575571    PK  PRIMARY_KEY_CONSTRAINT
2006-02-22 07:01:47.840 2006-02-22 07:01:47.840 0       0      0
UPKCL_sales
261575970  NULL     1      245575913    PK  PRIMARY_KEY_CONSTRAINT
2006-02-22 07:01:47.857 2006-02-22 07:01:47.857 0       0      0
UPK_storeid
229575856  NULL     1      213575799    PK  PRIMARY_KEY_CONSTRAINT
2006-02-22 07:01:47.840 2006-02-22 07:01:47.840 0       0      0
UPKCL_pubind
21575115  NULL     1      5575058      PK  PRIMARY_KEY_CONSTRAINT
2006-02-22 07:01:47.827 2006-02-22 07:01:47.827 0       0      0
UPKCL_titleidind
85575343  NULL     1      69575286      PK  PRIMARY_KEY_CONSTRAINT
2006-02-22 07:01:47.840 2006-02-22 07:01:47.840 0       0      0
sysrowsetcolumns
4       NULL     4      0         S  SYSTEM_TABLE
2005-10-14 01:36:15.923 2005-10-14 01:36:15.923 1       0      0
sysrowsets
5       NULL     4      0         S  SYSTEM_TABLE
2005-10-14 01:36:15.910 2005-10-14 01:36:15.910 1       0      0
sysallocunits
7       NULL     4      0         S  SYSTEM_TABLE
2005-10-14 01:36:15.910 2005-10-14 01:36:15.910 1       0      0
sysfiles1
8       NULL     4      0         S  SYSTEM_TABLE
2003-04-08 09:13:38.093 2003-04-08 09:13:38.093 1       0      0
syshobtcolumns
13      NULL     4      0         S  SYSTEM_TABLE
2005-10-14 01:36:15.940 2005-10-14 01:36:15.940 1       0      0
syshobts
15      NULL     4      0         S  SYSTEM_TABLE
2005-10-14 01:36:15.923 2005-10-14 01:36:15.923 1       0      0
sysftinds
25      NULL     4      0         S  SYSTEM_TABLE
2005-10-14 01:36:17.063 2005-10-14 01:36:17.063 1       0      0
sysserefs
26      NULL     4      0         S  SYSTEM_TABLE
2005-10-14 01:36:15.940 2005-10-14 01:36:15.940 1       0      0
sysowners
27      NULL     4      0         S  SYSTEM_TABLE
2005-10-14 01:36:17.050 2005-10-14 01:36:17.050 1       0      0
sysprivs
29      NULL     4      0         S  SYSTEM_TABLE
2005-10-14 01:36:15.877 2005-10-14 01:36:15.877 1       0      0
```

```
sysschobjs
34      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:15.987 2005-10-14 01:36:15.987 1        0       0
syscolpars
41      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:17.017 2005-10-14 01:36:17.017 1        0       0
sysnsobjs
44      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:16.000 2005-10-14 01:36:16.000 1        0       0
syscerts
46      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:25.173 2005-10-14 01:36:25.193 1        0       0
sysxprops
49      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:18.063 2005-10-14 01:36:18.063 1        0       0
sysscalartypes
50      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:15.847 2005-10-14 01:36:15.847 1        0       0
systypedsubobjs
51      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:17.033 2005-10-14 01:36:17.033 1        0       0
sysidxstats
54      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:17.033 2005-10-14 01:36:17.033 1        0       0
sysiscols
55      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:17.050 2005-10-14 01:36:17.050 1        0       0
sysbinobjs
58      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:22.110 2005-10-14 01:36:22.123 1        0       0
sysobjvalues
60      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:15.970 2005-10-14 01:36:15.970 1        0       0
sysclsobjs
64      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:16.000 2005-10-14 01:36:16.000 1        0       0
sysrowsetrefs
65      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:17.050 2005-10-14 01:36:17.050 1        0       0
sysremsvcbinds
67      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:24.127 2005-10-14 01:36:24.143 1        0       0
sysxmitqueue
68      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:24.143 2005-10-14 01:36:24.153 1        0       0
sysrts
69      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:24.143 2005-10-14 01:36:24.160 1        0       0
sysconvgroup
71      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:24.127 2005-10-14 01:36:24.147 1        0       0
sysdesend
72      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:24.160 2005-10-14 01:36:25.160 1        0       0
```

```
sysdercv
73      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:25.157 2005-10-14 01:36:25.167 1        0       0
syssingleobjrefs
74      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:15.860 2005-10-14 01:36:15.860 1        0       0
sysmultiobjrefs
75      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:15.877 2005-10-14 01:36:15.877 1        0       0
sysdbfiles
76      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:15.953 2005-10-14 01:36:15.953 1        0       0
sysguidrefs
78      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:23.110 2005-10-14 01:36:23.123 1        0       0
sysqnames
90      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:23.110 2005-10-14 01:36:23.127 1        0       0
sysxmlcomponent
91      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:23.127 2005-10-14 01:36:23.140 1        0       0
sysxmlfacet
92      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:23.127 2005-10-14 01:36:23.137 1        0       0
sysxmlplacement
93      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:23.127 2005-10-14 01:36:24.137 1        0       0
sysobjkeycrypts
94      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:25.157 2005-10-14 01:36:25.177 1        0       0
sysasymkeys
95      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:25.173 2005-10-14 01:36:25.193 1        0       0
syssqlguides
96      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:25.190 2005-10-14 01:36:25.207 1        0       0
sysbinsubobjs
97      NULL      4       0          S   SYSTEM_TABLE
2005-10-14 01:36:22.110 2005-10-14 01:36:23.120 1        0       0
ServiceBrokerQueue
2041058307 NULL      1       0          SQ   SERVICE_QUEUE
2005-10-14 01:36:25.377 2005-10-14 01:36:25.377 1        0       0
QueryNotificationErrorsQueue
1977058079 NULL      1       0          SQ   SERVICE_QUEUE
2005-10-14 01:36:25.360 2005-10-14 01:36:25.360 1        0       0
EventNotificationErrorsQueue
2009058193 NULL      1       0          SQ   SERVICE_QUEUE
2005-10-14 01:36:25.377 2005-10-14 01:36:25.377 1        0       0
employee_insupd
645577338  NULL      1       501576825     TR   SQL_TRIGGER
2006-02-22 07:01:47.903 2006-02-22 07:01:47.903 0        0       0
authors
2073058421 NULL      1       0          U   USER_TABLE
2006-02-22 07:01:47.717 2006-02-22 07:01:49.997 0        0       0
publishers
```

```
5575058    NULL      1       0         U  USER_TABLE
2006-02-22 07:01:47.793 2006-02-22 07:01:47.890 0       0       0
titles
69575286   NULL      1       0         U  USER_TABLE
2006-02-22 07:01:47.840 2006-02-22 07:01:50.153 0       0       0
titleauthor
149575571   NULL      1       0         U  USER_TABLE
2006-02-22 07:01:47.840 2006-02-22 07:01:50.263 0       0       0
sales
245575913   NULL      1       0         U  USER_TABLE
2006-02-22 07:01:47.857 2006-02-22 07:01:50.090 0       0       0
stores
213575799   NULL      1       0         U  USER_TABLE
2006-02-22 07:01:47.840 2006-02-22 07:01:47.860 0       0       0
roysched
309576141   NULL      1       0         U  USER_TABLE
2006-02-22 07:01:47.857 2006-02-22 07:01:50.343 0       0       0
pub_info
453576654   NULL      1       0         U  USER_TABLE
2006-02-22 07:01:47.873 2006-02-22 07:01:47.877 0       0       0
jobs
373576369   NULL      1       0         U  USER_TABLE
2006-02-22 07:01:47.857 2006-02-22 07:01:47.890 0       0       0
discounts
341576255   NULL      1       0         U  USER_TABLE
2006-02-22 07:01:47.857 2006-02-22 07:01:47.860 0       0       0
employee
501576825   NULL      1       0         U  USER_TABLE
2006-02-22 07:01:47.873 2006-02-22 07:01:49.903 0       0       0
titleview
661577395   NULL      1       0         V  VIEW
2006-02-22 07:01:50.373 2006-02-22 07:01:50.373 0       0       0
```

Notice all the rows with this type of SYSTEM_TABLE. But earlier I said you cannot read system tables. This is true, but you can see a list of all of them. These system tables are the base system tables stored in your database. Earlier beta releases did not expose these table names in this view, but the decision was made to show them primarily because the object ID values were showing up in error messages and users were confused when they tried to figure out what table was associated with the message.

If you tried to query one of these tables like this:

```
select * from sysschobjs
```

you would get the following error:

```
Msg 208, Level 16, State 1, Line 1
Invalid object name 'sysschobjs'.
```

What about the executable system objects stored in the `resource` database? If I cannot actually access the `resource` database directly, how do I see a list of these object? Well, SQL Server includes a catalog view just for this purpose:

```
select * from sys.system_objects
name
object_id  principal_id schema_id  parent_object_id type type_desc
create_date       modify_date       is_ms_shipped is_published is_schema_published
------------------------------------------------------------------------------------
----------------------------------------- ----------- ------------ ----------- --
-------------- ---- -----------------------------------------------------------
---------------------- ---------------------- ------------ ----------- ---------
----------
fn_cColvEntries_80
-61545096  NULL      4       0         FN  SQL_SCALAR_FUNCTION
2006-02-17 14:43:52.820 2006-02-17 14:43:52.820 1        0       0
fn_fIsColTracked
-242919846 NULL      4       0         FN  SQL_SCALAR_FUNCTION
2006-02-17 14:43:53.040 2006-02-17 14:43:53.040 1        0       0
fn_GetCurrentPrincipal
-986367695 NULL      4       0         FN  SQL_SCALAR_FUNCTION
2006-02-17 14:43:46.130 2006-02-17 14:43:46.130 1        0       0
fn_GetRowsetIdFromRowDump
-633331936 NULL      4       0         FN  SQL_SCALAR_FUNCTION
2006-02-17 14:41:07.853 2006-02-17 14:41:07.853 1        0       0
fn_IsBitSetInBitmask
-92987834  NULL      4       0         FN  SQL_SCALAR_FUNCTION
2006-02-17 14:44:11.087 2006-02-17 14:44:11.087 1        0       0
fn_isrolemember
-977642094 NULL      4       0         FN  SQL_SCALAR_FUNCTION
2006-02-17 14:44:15.147 2006-02-17 14:44:15.147 1        0       0
.
.
sp_help
-784136858 NULL      4       0         P   SQL_STORED_PROCEDURE
2006-02-17 14:41:29.803 2006-02-17 14:41:29.803 1        0       0
sp_help_agent_default
-930237674 NULL      4       0         P   SQL_STORED_PROCEDURE
2006-02-17 14:46:15.130 2006-02-17 14:46:15.130 1        0       0
sp_help_agent_parameter
-244913556 NULL      4       0         P   SQL_STORED_PROCEDURE
2006-02-17 14:46:16.443 2006-02-17 14:46:16.443 1        0       0
sp_help_agent_profile
-463476349 NULL      4       0         P   SQL_STORED_PROCEDURE
2006-02-17 14:46:14.913 2006-02-17 14:46:14.913 1        0       0
sp_help_datatype_mapping
-721021307 NULL      4       0         P   SQL_STORED_PROCEDURE
2006-02-17 14:46:22.350 2006-02-17 14:46:22.350 1        0       0
.
.
.
indexes
-397    NULL      4       0         V   VIEW
2006-02-17 14:38:43.433 2006-02-17 14:38:43.433 1        0       0
```

```
internal_tables
-468     NULL      4       0          V   VIEW
2006-02-17 14:38:42.340 2006-02-17 14:38:42.340 1        0       0
KEY_COLUMN_USAGE
-784887024 NULL      3       0          V   VIEW
2006-02-17 14:43:25.880 2006-02-17 14:43:25.880 1        0       0
key_constraints
-406     NULL      4       0          V   VIEW
2006-02-17 14:38:46.057 2006-02-17 14:38:46.057 1        0       0
key_encryptions
-465     NULL      4       0          V   VIEW
2006-02-17 14:39:26.200 2006-02-17 14:39:26.200 1        0       0
linked_logins
-222     NULL      4       0          V   VIEW
2006-02-17 14:39:17.120 2006-02-17 14:39:17.120 1        0       0
login_token
-77438453  NULL      4       0          V   VIEW
2006-02-17 14:39:49.777 2006-02-17 14:39:49.777 1        0       0
master_files
-216     NULL      4       0          V   VIEW
2006-02-17 14:39:11.870 2006-02-17 14:39:11.870 1        0       0
master_key_passwords
-243     NULL      4       0          V   VIEW
2006-02-17 14:39:14.167 2006-02-17 14:39:14.167 1        0       0
message_type_xml_schema_collection_usages
-478     NULL      4       0          V   VIEW
2006-02-17 14:38:59.730 2006-02-17 14:38:59.730 1        0       0
messages
-225     NULL      4       0          V   VIEW
2006-02-17 14:38:57.980 2006-02-17 14:38:57.980 1        0       0
module_assembly_usages
-484     NULL      4       0          V   VIEW
2006-02-17 14:38:51.307 2006-02-17 14:38:51.307 1        0       0
numbered_procedure_parameters
-419     NULL      4       0          V   VIEW
2006-02-17 14:38:52.510 2006-02-17 14:38:52.510 1        0       0
numbered_procedures
-418     NULL      4       0          V   VIEW
2006-02-17 14:38:52.183 2006-02-17 14:38:52.183 1        0       0
objects
-385     NULL      4       0          V   VIEW
2006-02-17 14:38:36.870 2006-02-17 14:38:36.870 1        0       0
.
.
.
sysindexes
-134     NULL      4       0          V   VIEW
2006-02-17 14:39:33.090 2006-02-17 14:39:33.090 1        0       0
sysindexkeys
-135     NULL      4       0          V   VIEW
2006-02-17 14:39:33.417 2006-02-17 14:39:33.417 1        0       0
syslanguages
-194     NULL      4       0          V   VIEW
2006-02-17 14:39:26.853 2006-02-17 14:39:26.853 1        0       0
```

```
syslockinfo
-204    NULL     4      0         V  VIEW
2006-02-17 14:41:00.633 2006-02-17 14:41:00.633 1        0       0
syslogins
-205    NULL     4      0         V  VIEW
2006-02-17 14:39:37.463 2006-02-17 14:39:37.463 1        0       0
sysmembers
-141    NULL     4      0         V  VIEW
2006-02-17 14:39:34.400 2006-02-17 14:39:34.400 1        0       0
sysmessages
-206    NULL     4      0         V  VIEW
2006-02-17 14:39:38.777 2006-02-17 14:39:38.777 1        0       0
sysobjects
-105    NULL     4      0         V  VIEW
2006-02-17 14:38:39.823 2006-02-17 14:38:39.823 1        0       0
.
.
.
xp_adsirequest
-60872162  NULL     4      0         X  EXTENDED_STORED_PROCEDURE
2006-02-17 14:53:06.237 2006-02-17 14:53:06.237 1        0       0
xp_availablemedia
-196500590 NULL     4      0         X  EXTENDED_STORED_PROCEDURE
2006-02-17 14:53:06.033 2006-02-17 14:53:06.033 1        0       0
xp_cleanupwebtask
-113576977 NULL     4      0         X  EXTENDED_STORED_PROCEDURE
2006-02-17 14:53:20.300 2006-02-17 14:53:20.300 1        0       0
xp_cmdshell
-1008137134 NULL     4      0         X  EXTENDED_STORED_PROCEDURE
2006-02-17 14:41:25.383 2006-02-17 14:41:25.383 1        0       0
```

This is not a complete list, because more than 1,700 system objects are accessible via the sys schema. You will see in this list some familiar names that used to be stored in the master database in SQL Server 2000, such as `sp_help`. You will also see new names such as views called `indexes`, `master_files`, and `objects`. Notice that names like `sysindexes` and `sysobjects` have a type `VIEW`. Also, you can see at the end of this list that Microsoft internal extended stored procedures are stored in the `resource` database.

Notice something interesting about the `object_id` of these names. They are listed as values `< 0`. Whenever you see an `object_id` listed as `< 0`, you know it is a system object that comes from the `resource` database.

I reference many of the catalog views in the remaining sections of this chapter. SQL Server Books Online has a complete reference of all the views, including the descripton of the column definitions.

One specific catalog view that I should mention is `sys.system_sql_modules`. In the early versions of the beta, there was no method to see the text of any system object. Based on user input during the beta, the SQL Server development team chose to expose the definitions of these objects with `sys.system_sql_modules`. The following syntax:

```
select * from sys.system_sql_modules where object_id = object_id('sys.sysobjects')
```

shows you the definition of the sysobjects catalog view. If you run this query, you will see it references the sys.sysschobjs base system table. Be careful relying on the results of this view. The SQL Server development team exposed this information so that you can see how views, procedures, and so on were built. But you cannot change the text, and the development team certainly could change these over time as they fix bugs, enhance performance, or change the design of base system tables. To help give you a quick start on catalog views, Table 2-1 shows a comparison of what catalog view to use based on the SQL 2000 equivalent system table.

TABLE 2-1 What catalog view to use based on the SQL 2000 equivalent system table

SQL Server 2000 System Table	SQL Server 2005 Catalog View
`sysobjects`	`sys.objects`
`sysindexes`	`sys.indexes+sys.partitions +sys.allocation_units`
`syscolumns`	`sys.columns`
`syscomments`	`sys.sql_modules`
`sysdatabases`	`sys.databases`
`sysfiles`	`sys.database_files`
`sysaltfiles`	`sys.master_files`

Let me make one final comment about base system tables, catalog views, and the `resource` database. Base system tables exist in every database and store all the metadata. All the catalog views and executable system objects (sometimes called just system objects in Books Online) are not stored in your database. The definition of these objects exists only in the `resource` database. The `resource` database is a *database*, so it has base system tables. Think of it this way. When you create a view in your database, SQL Server stores the definition of this view in base system tables in your database. The `resource` database is simply preloaded with a bunch of views, procedures, and so forth. No *user tables* exist in the `resource` database.

Allocation Structures

The internal structures used to organize allocation for SQL Server 2005 have not changed dramatically. Concepts such as `GAM`, `IAM`, `SGAM`, and `PFS` all still exist to internally track and organize allocation of pages and extents.

There are some differences at a higher level of allocation. The first difference is support for partitions. SQL Server 2000 enables you to place tables or indexes on specific disks by using filegroups. SQL Server 2005 expands this capability to place horizontal slices of tables or indexes on specific disks using partitions. Partitions allow you to specify that a particular range of values within a table or index is stored on specific filegroups. Every table or index has at least one partition, as can be seen by querying the `sys.partitions` catalog view. Each partition can have up to three different allocation units. An allocation unit is equivalent to an `IAM` chain. In SQL Server 2000, a table could have two different `IAM` chains: one for the data and one for `TEXT/IMAGE` data. SQL Server 2005 supports three different IAM chains for an object: data, LOB (TEXT/IMAGE), and SLOB. A Small Large Binary Object (SLOB) is also referred to as row overflow data. SQL Server 2005 enables you to store data in a row that is larger than the SQL Server page size (8KB). This is done by supporting a different IAM chain to store this extended row data.

Figure 2-2 shows the allocation structure objects and corresponding catalog views.

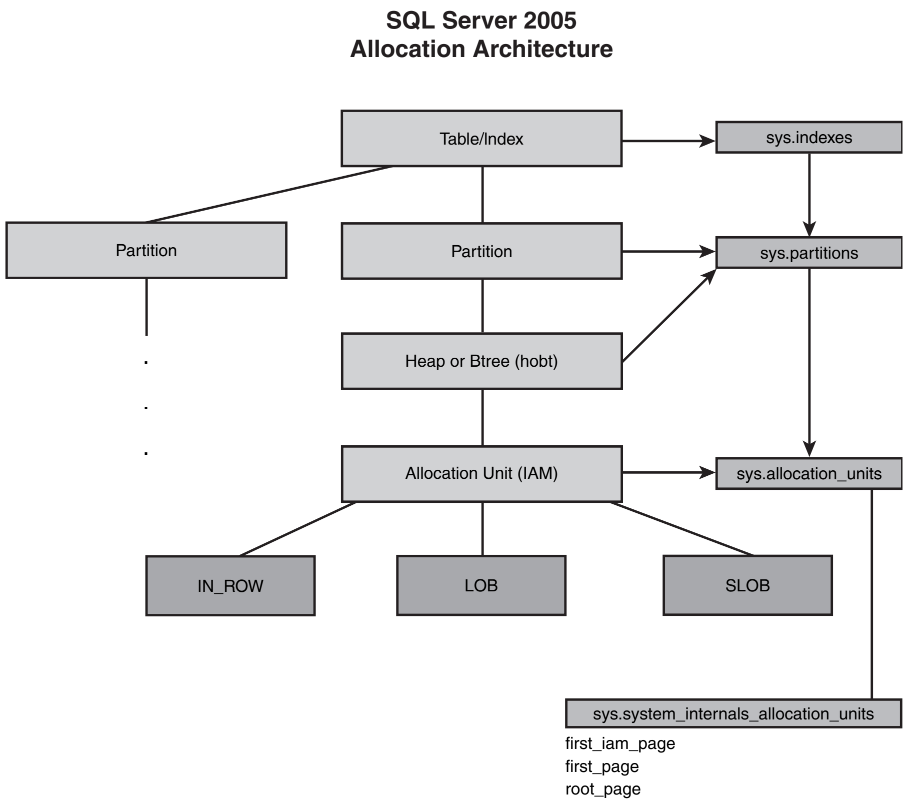


FIGURE 2-2 Allocation structure objects and corresponding catalog views

Let's take a look at a few of the catalog views that describe allocation structures. Let's create a table and index using the following query:

```
create table test_table (col1 int primary key clustered, col2 int,
col3 text, col4 varchar(5000) not null, col5 varchar(5000) not null)
go
create index test_idx on test_table (col2)
go
insert into test_table values (1, 1, 'this is text', 'this is test', 'this is
test')
go
```

To see the organization of allocation for this table and index, first look at the sys.partitions catalog view:

```
select * from sys.partitions where object_id = object_id('test_table')
partition_id      object_id   index_id   partition_number hobt_id        rows
------------------- ----------- ----------- --------------- ------------------- -
-----
72057594038583296   2137058649 1      1         72057594038583296   1
72057594038648832   2137058649 2       1          72057594038648832   1
```

As you can see, the table and nonclustered index each have one partition. Notice the second column from the right, hobt_id. Hobt stands for heap or b-tree. Each partition is made up of exactly one heap or b-tree index. Notice the value of hobt_id is the same as partition_id. Why is the column needed? Very early designs of SQL Server 2005 included the ability for a heap or b-tree to store rows from multiple partitions. In other words, a partition could have its data spread across multiple allocation units of the same type. The final version of SQL Server 2005 doesn't contain this capability, but the base system tables still support this concept. (This is why you see in the list of tables the name syshobts.) In SQL Server 2005, a partition and hobt are basically equivalent, which is why the ID for both is the same. You can see a reflection of this design in the page header of a database page. The following is part of the output for DBCC PAGE for a page in a table I created in the master database:

```
PAGE HEADER:
Page @0x0386A000
m_pageId = (1:509)           m_headerVersion = 1          m_type = 1
m_typeFlagBits = 0x4          m_level = 0             m_flagBits = 0xe200
m_objId (AllocUnitId.idObj) = 87   m_indexId (AllocUnitId.idInd) = 256
Metadata: AllocUnitId = 72057594043629568
Metadata: PartitionId = 72057594038779904                Metadata: IndexId = 1
Metadata: ObjectId = 1291151645   m_prevPage = (0:0)          m_nextPage = (0:0)
pminlen = 12              m_slotCnt = 1            m_freeCnt = 8017
m_freeData = 173            m_reservedCnt = 0          m_lsn = (2006:16:1)
m_xactReserved = 0           m_xdesId = (0:0)           m_ghostRecCnt = 0
m_tornBits = -1570474108
```

Instead of storing the object_id and index_id values as SQL Server did in previous versions, it stores the allocation_unit_id as seen in the sys.allocation_units catalog view. DBCC PAGE decodes the allocation_unit id value such that it includes the object_id and index_id from the system catalog metadata. You can see this from the areas I highlighted that start with the word *metadata*.

Let's now look at the allocation units for the table and index using the sys.allocation_units catalog view:

```
-- The table has a text column and possible row overflow ---- data so there
should be 3 rows for the clustered index --- and 1 row for the ncl index
select object_name(pt.object_id), pt.index_id, au.* from sys.allocation_units au
```

```
join sys.partitions pt
on au.container_id = pt.partition_id
and pt.object_id = object_id('test_table')
order by container_id
name
index_id  allocation_unit_id  type type_desc
container_id     data_space_id total_pages     used_pages      data_pages
---------------------------------------------------------------------------------
--------------------------------------- ---------- ------------------ ---- -
------------------------------------------------------- ------------------
------------ ------------------- ------------------- -------------------
test_table
1      72057594042843136  1  IN_ROW_DATA
72057594038583296  1       2           2           1
test_table
1      72057594042908672  3  ROW_OVERFLOW_DATA
72057594038583296  1       0           0           0
test_table
1      72057594042974208  2  LOB_DATA
72057594038583296  1       2           2           0
test_table
2      72057594043039744  1  IN_ROW_DATA
72057594038648832  1       2           2           1
```

Notice that `index_id = 1` (the clustered index for `test_table`) has three types: `IN_ROW_DATA` (the actual data rows), `ROW_OVERFLOW_DATA` (this is the SLOB because we created the table with variable-length character columns that exceed 8KB), and `LOB_DATA` (the `TEXT/IMAGE` data). `index_id = 2`, the nonclustered index, has one allocation unit. Notice in the preceding query the column to join back to `sys.partitions` is `container_id`. Because of what I described with hobts, you could join to either `partition_id` or `hobt_id`. If you are a veteran SQL Server support engineer like me, you might see that something is missing from `sys.allocation_units`. What about `first`, `root`, and `first_IAM`? Initially, the SQL Server development team did not think anyone needed to see these columns. Feedback to Microsoft was that it would be valuable to still see these values. So, late in the beta cycle, the `sys.system_internals_allocation_units` catalog view was created. Let's run the preceding query again, this time changing it to use this view:

```
-- IAM information is only found in this internal catalog --- view. This view
may change so could
-- break your application if you rely on the result set
select object_name(pt.object_id) as name, pt.index_id, sau.* from
sys.system_internals_allocation_units sau
join sys.partitions pt
on sau.container_id = pt.partition_id
and pt.object_id = object_id('test_table')
order by container_id
go
name                                                                 index_id
allocation_unit_id  type type_desc                            container_id
filegroup_id total_pages     used_pages      data_pages      first_page
root_page   first_iam_page
```

```
----------------------------------------------------------------------------
---------------------------------------- ---------- ------------------ ---- -
------------------------------------------------------- ------------------
----------- ------------------ ------------------ ------------------ -------
------ ------------- -------------
test_table                                                              1
72057594042843136  1  IN_ROW_DATA                         72057594038583296  1
2          2          1          0x9A0000000100 0x9A0000000100 0x9B0000000100
test_table                                                              1
72057594042908672  3  ROW_OVERFLOW_DATA                    72057594038583296  1
0          0          0          0x000000000000 0x000000000000 0x000000000000
test_table                                                              1
72057594042974208  2  LOB_DATA                            72057594038583296  1
2          2          0          0x980000000100 0x980000000100 0x990000000100
test_table                                                                  2
72057594043039744  1  IN_ROW_DATA                          72057594038648832
1       2          2          1          0x9C0000000100 0x9C0000000100
0x9D0000000100
```

Now the same page numbers in hex as you could find from sysindexes in SQL Server 2000 are available. Be careful with the usage of this view. As mentioned in the documentation, it might change over time, depending on changes to the architecture for allocation.

Database Checksum

One of the challenges in PSS for customer cases involving database corruption is to pinpoint whether the problem is specifically caused by a hardware or system problem. SQL Server 2005 adds checksum capabilities for database pages and log blocks. This is an important feature to determine how a database page might have become damaged. Here is how it works.

If you enable a database for checksum (it is on by default, or you enable it with ALTER DATABASE), right before the engine writes a page to disk, it calculates a checksum value based on the bits on the page and writes this value into the page header. When the page is read back from disk, SQL Server calculates the checksum value based on bits on the page (excluding the checksum itself in the header) and compares that to the checksum value in the page header. If they do not match, SQL Server knows the page must have been altered after it submitted it to be written to disk. The server doesn't know who altered it, but the path of IO from the operating system to the disk is subject to scrutiny.

Fast Recovery

A number of customer cases come into PSS in which the customer is struggling because the database is not accessible because of the length of database recovery on startup. Database recovery has three basic phases: analysis, redo, and undo. The final phase, undo, typically takes the longest and is the source of pain. SQL Server 2005 takes a step forward in database availability by allowing access to the database during the undo phase of recovery. How is this accomplished? The engine during recovery acquires locks during the redo phase and holds these locks for any uncommitted transaction. When the undo phase begins, recovery is in effect a "user" who is running transactions to undo operations that need to be rolled back. Because locks are acquired to perform this operation, transaction consistency can be maintained. Users can now access the database and run standard transactions. The caveat is that users may get blocked by a system session that is running recovery (because it is holding locks while processing undo). Users who need to access data not related to any transactions being rolled back by recovery will not be affected. The restrictions on this feature are as follows:

- It is supported only for Enterprise Edition.
- It works only for databases set up for full recovery.
- It works only for crash recovery. (In other words, it does not work when restoring a database.)

Deferred Transactions

Another important enhancement to database recovery is the concept of deferred transactions. Consider this scenario. Your system has an unexpected crash and reboots. SQL Server restarts and now has to run recovery on all databases. Because of a problem with your disk system, one of the database pages needed by recovery is damaged. Because the database is enabled for checksum, recovery detects a checksum error. In SQL Server 2000, a damaged page detected during recovery would result in an instant `SUSPECT` database. The entire database would be inaccessible, and the only method to recover reliably would be to restore the entire database from backup. SQL Server addresses this with a clever solution. In situations where damage to a single page is detected during recovery, the page is marked with a special bit called *RestorePending* so that it cannot be accessed. Furthermore, the transaction associated with the page is "deferred" if it is an active transaction that must be rolled back. What this means is that locks associated with the transaction are held after redo and are not released after undo. In fact, undo for the transaction is skipped.

So at the time recovery is complete, the database is still online, the page that is damaged is not accessible, and any locks associated with an associated uncommitted transaction are held. But the database status remains `ONLINE` and not `SUSPECT`. So, users can access the database but will encounter an error when accessing the page. Also, if the transaction associated with the page is deferred, a user could be blocked. (The locks are held by `session_id = -3`.) Resolution to the problem could then be to just restore the damaged page. More on the details behind this when I talk about recovering an inaccessible database.

Deferred transactions are supported for both crash and restore recovery, but this does not work for recovery when a database is attached.

Read-Only Compressed Databases

A frequent question I see about databases is whether you can use NTFS compression for the database and log files. The largest concern is consistency, because the server cannot guarantee sector-aligned writes. However, a read-only database has no consistency concern. Therefore, the SQL Server development team decided to allow a read-only database to be stored on NTFS compressed files. SQL Server 2000 did not prevent you from creating a read-write database on an NTFS compressed volume, even though this is not a supported configuration. However, in SQL Server 2005, if you shut down SQL Server, compress the files of a read-write database, and then try to restart SQL Server, you see the following errors in the ERRORLOG:

```
2006-03-18 22:49:14.87 spid15s   Starting up database 'pubs'.
2006-03-18 22:49:15.01 spid15s   Error: 5118, Severity: 16, State: 1.
2006-03-18 22:49:15.01 spid15s   The file  "C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\DATA\pubs.mdf" is compressed but does not reside in a
read-only database or filegroup. The file must be decompressed.
```

As the message implies, it is also possible to mark a filegroup as read-only and compress the files just for that filegroup even though the rest of the database is read-write.

SQL Server 2005 Enhancements

I have discussed several storage internal concepts, including several enhancements for SQL Server 2005 based on customer and PSS experience with data recovery in SQL Server 2000 and previous versions.

A discussion of new enhancements for SQL Server 2005 data recovery would not be complete without talking about BACKUP/RESTORE and DBCC CHECKDB.

BACKUP Enhancements

The most noteworthy enhancement to BACKUP is verification. If you execute the T-SQL BACKUP command using the WITH CHECKSUM option, two things happen:

- As each database page is read from the database file from disk, the server verifies the checksum value in the page header if it exists. If it fails, the server raises an error and stops the backup.
- The server calculates a checksum value for all the bits from all pages in the backup and writes this in the backup media. This is called the backup checksum.

These both serve important purposes. By verifying the checksum on pages before the server writes them to the backup, you can prevent the server writing bad pages in your backup. Second, by writing a checksum value for the entire backup stream, the server can verify using RESTORE whether any bits in the backup media itself were damaged after backup writes were submitted to the target media. It is important to know that no new checksum value is written during BACKUP. So, if the database page did not contain a checksum value, a new one is not calculated and then written to the page before it is written to the backup media. But this page is still used as part of calculating the backup checksum.

If you do encounter an error when using WITH CHECKSUM because of a damaged page, you can choose to ignore any error using the WITH CONTINUE_AFTER_ERROR option. The best course of action is to resolve the checksum problem before taking the backup; in an emergency situation to ensure the rest of the database is backed up, however, you might consider using this option.

One other nice addition for SQL Server 2005 `BACKUP` is full-text data. Previously, to back up your full-text data, you had to manually back up the full-text catalog files associated with the database. Now if your database has full-text data, SQL Server automatically includes the full-text catalog information into the backup information associated with the database. You can even just back up the full-text catalog on it own using the T-SQL `BACKUP` command. (Refer to the Books Online reference for `BACKUP` for the proper syntax.)

RESTORE Enhancements

Until SQL Server 2005, the `RESTORE VERIFYONLY` command was not all that useful. This is because this option did not really verify much of what was contained in the backup set. That has changed for SQL Server 2005 in two ways:

- If the backup media contains a backup checksum, by default the server verifies it. The server also verifies any checksum that exists on a given page in the backup.
- Even if the backup does not contain any checksum, the server verifies that the page ID in the page header is valid.

The same verification occurs by default for the standard `RESTORE` command, not just with `VERIFYONLY`. But `VERIFYONLY` is now a useful command in your arsenal. With checksums, it becomes a quick solution to verifying the integrity of the backup media. (Remember, however, this doesn't mean the backup will actually restore successfully, because recovery could still fail.)

Another option for the `RESTORE` command that will be useful in emergency situations is `CONTINUE_AFTER_ERROR`. Before this option was added and a customer encountered a corrupt backup, there was no way to know whether a single page was damaged or whether the entire backup was bad.

If you encounter an error with `RESTORE` (such as failure verifying checksum) on SQL Server 2005, you can use the `CONTINUE_AFTER_ERROR` option. The engine just ignores all errors and loads all pages in the database as they exist in the backup. You can then decide how to fix any remaining errors (for example, using `DBCC CHECKDB` with repair options). This is not an option you want to use on a regular basis. If you do, it means you have problems with your backups. However, it might come in handy on some late night when you do have bad backups but want to extract as much as you can from them.

The last important enhancement for `RESTORE` is the page-level restore. A `PAGE` option has been added to the `RESTORE` command to specify page numbers (maximum of 1,000). The addition of this option is yet another to help increase database availability. Consider a case where you encounter a checksum error on a single page for a 10TB database. If you want to restore just this page, you can from the latest full backup and restore subsequent log backups to make the transactions affecting the page consistent. And with Enterprise Edition, you can do all of this online, allowing other users to access unaffected areas of the database. Of course, the speed of the RESTORE alone is improved, because the server only has to write one or more pages from the backup to the database. (It still takes time to find and read the page from the backup.) I fully expect customers to take advantage of this option, especially those who call PSS and want us to help with some type of advanced recovery because they cannot afford the downtime of a full restore. There is an exercise at the end of chapter you can use to see how to use page-level restore after encountering a damaged page.

DBCC CHECKDB Enhancements

Another area of significant investment for SQL Server 2005 is DBCC CHECKDB. A former PSS colleague of mine, Ryan Stonecipher, now owns this code. He worked with Paul Randal, who used to own it, to incorporate feedback from customers and PSS throughout the beta. The result is some of the following feature enhancements.

DATA_PURITY

DBCC CHECKDB can verify all types of things. But in previous releases of SQL Server, one thing it did not verify was the validity of values within certain data types (such as date/time). The SQL Server development team actually added an undocumented trace flag in SQL Server 2000 to check this because a few customers reported databases with invalid datetime or decimal data within the column values. This led to some strange problems including what appeared to be incorrect results. So, the development team decided in SQL Server 2005 to just add an option to check datetime and decimal columns for valid ranges of values as specified for that type.

For a new SQL Server 2005 database, DATA_PURITY checks are on by default (unless you use the WITH PHYSICAL_ONLY option). For an upgraded database, you must use WITH DATA_PURITY one time, and then it is implied from that point forward.

Progress Reporting

One common question for users of DBCC CHECKDB is a bit like your kids asking you in the car, “When do we get there?” Customers call PSS after DBCC CHECKDB has been running for an hour and say, “When will it be done?” So, the SQL Server development team added progress reporting capabilities for CHECKDB. A user can query the percent_complete and command columns of the sys.dm_exec_requests dynamic management view to see the current progress of CHECKDB. The command column displays a set of predefined values that describe the phase of execution of CHECKDB. (For example, DBCC SYS CHECK means CHECKDB is checking the consistency of system tables. The percent_complete column marks the progress within this phase.) SQL Server Books Online has a complete description of the phases and whether progress is reported for that phase.

“Last Known Good”

One of the questions I have asked customers when investigating a case involving database corruption is this: “When was the last time DBCC CHECKDB reported no errors for this database?” If the customer did not save all CHECKDB results or if the ERRORLOG files have wrapped, there is no way to answer this question. SQL Server 2005 saves in the database information about the last time a DBCC CHECKDB was run without errors on the database.

Anytime the database is started, the information about the "last known good" clean DBCC is reported in the ERRORLOG like the following:

```
2005-09-22 11:56:48.42 Server   Database mirroring has been enabled on this
instance of SQL Server.
2005-09-22 11:56:48.42 spid5s   Starting up database 'master'.
2005-09-22 11:56:48.73 spid5s   Recovery is writing a checkpoint in database
'master' (1). This is an informational message only. No user action is required.
2005-09-22 11:56:48.84 spid5s   CHECKDB for database 'master' finished without
errors on 2005-09-22 11:31:45.990 (local time). This is an informational
message only; no user action is required.
```

As you can see in this message, a CHECKDB was deemed to be "good." This information is updated every time a CHECKDB completes without errors for a specific database.

In addition to saving and recording the last clean CHECKDB, the SQL Server development team also enhanced the report in the ERRORLOG for each CHECKDB execution. The server now includes a duration value, so you can see how long it typically takes to run CHECKDB for your database. Here is an example of this ERRORLOG output:

```
2006-03-19 19:35:40.15 spid51   DBCC CHECKDB (troy) executed by NORTHAMERICA
\bobward found 0 errors and repaired 0 errors. Elapsed time: 0 hours 0 minutes 2
seconds.
```

Online Uses Database Snapshot

In SQL Server 2000, *online* DBCC CHECKDB reads the transaction log to check the consistency of the database. By online, I mean that the database is in MULTI_USER mode. Although this technique works, in some cases this might cause false CHECKDB errors. Therefore, in SQL Server 2005, CHECKDB takes advantage of the new snapshot database feature. An online CHECKDB now creates a database snapshot of the current database and uses the snapshot to check database consistency. This now makes online CHECKDB extremely simple. Just run the consistency check on the snapshot and you are guaranteed a consistent set of pages at the point in time CHECKDB was run. If a snapshot cannot be created (for example, because the databases are stored on the FAT file system), the table locks are used to ensure consistency.

Enhanced *CHECKCATLOG* Integrated

Any veteran SQL Server support engineer knows that DBCC CHECKCATALOG is not really worth running. In SQL Server 2000 and previous versions, only a few system tables were actually included in this check. Furthermore, the number of times an actual system table referential integrity occurs is few. Along with the new system table architecture, the SQL Server development team decided to actually implement a full catalog consistency check and include it by default when DBCC CHECKDB runs. DBCC CHECKCATALOG can still be run independently, but it is a quick operation as part of the overall CHECKDB execution.

Emergency Mode Repair

In SQL Server 2000, if the database is marked `SUSPECT` and you do not have a backup to restore, your options are pretty limited. One option some customers have chosen is to call Microsoft PSS to see whether they can help repair the database. This procedure involves some advanced undocumented commands that could end up resulting in a rebuild of the transaction log. This procedure can result in a `CHECKDB` with no errors, but logical consistency is now compromised. This request has come in so often that the SQL Server development team decided to include a recovery feature in the product so that customers could perform this operation themselves.

Data Recovery Best Practices

Before I dig into troubleshooting scenarios for various types of database recovery problems, let me share with you perspective on best practices for you to use to avoid problems. This section is not a complete study of disaster recovery strategies, but I give you some information you might find helpful based on my experience with data recovery customer cases.

BACKUP/RESTORE Best Practices

Your database is only as good as your last backup.

If you have a great backup strategy, more power to you. If you have any doubts, read on. You would be surprised how many customers call Microsoft PSS and have not backed up their database (or don't have a recent backup). If I had responsibility for any database being used in a business, my number one priority would be the safety and security of that data, which includes disaster recovery situations. I have many non-computer-savvy friends who I talk to about the files on their computer. They ask me how often they should back them up. I always tell them, "Your data is only as good as your most recent backup." As good as computer hardware is today, I've seen too many situations where someone didn't have a recent backup and lost data. It just takes one time of losing key data to learn to have timely and good backups. Don't let yourself run into that situation. You could run for years without ever needing a backup, but when you really need it, you will be so thankful that you spent careful time and consideration ensuring your backups are recent and up-to-date based on your needs and requirements.

Use the `BACKUP CHECKSUM` feature.

I personally would take any hit in backup performance that you may encounter when using the `WITH CHECKSUM` option of `BACKUP`. There is simply no better method in the database engine to detect whether the pages written to the backup are valid and to verify that the backup medium has not been damaged or altered after it is written. This feature actually makes `RESTORE VERIFYONLY` a viable option to determine whether you have a reasonable chance of restoring the database.

`RESTORE` is the only guaranteed method to verify backups.

Even with the checksum feature, if you want a guarantee that a backup can be verified, the only way is to restore it. This is because even if you use the `WITH CHECKSUM` feature, other problems could occur. For example, what if there is some logical problem with transaction log records so that when recovery runs during restore, a failure occurs? If the log record is physically correct but some logical problem exists that prevents redoing that log record within the engine, restoring the backup is the only way to find out.

As part of my advice on restore, I think it is important to test your disaster recovery strategy. For example, how long does it take to restore your backups at any given day or time of day? If your business requires the database to be back and available within 30 minutes during the day Monday through Friday, but with your current disaster recovery strategy, it would take 4 hours to restore your backups on Thursday at 2 p.m., clearly you have a problem. You may need to seek high-availability solutions such as database mirroring or log shipping.

Avoid network drive backups.

Customers seem to encounter more problems than not when backing up their databases or logs to a network drive. I've seen everything from damaged backups to network errors during the backup to Windows errors due to insufficient kernel resources when customers back up large databases to network drives. Using a storage area network (SAN) system seems to be fairly reliable, but I personally don't recommend you back up your database to a network mapped drive on another server on your network. If you have to store your backups on another server and you don't have a SAN system, I recommend you try to back up the database to a local drive, use a program or utility to compress the file (they usually put a checksum on the file as part of this), and then copy this file over the network to the remote server. If the compression utility supports checksum features, such as a cyclic redundancy check (CRC), when you uncompress you can at least safely know the bytes were copied to the remote server. Another alternative is to use a vendor backup solution that uses SQL Server VDI to stream the backup remotely to another computer.

Don't forget about page-level restore.

Remember when making key decision about restoring a database that you have a new feature to reduce the time to restore a backup sequence called page-level restore. Of course, to use this feature you need to know what page to restore, but in some cases, you may have damage to a single page for a multiterabyte database. If you have Enterprise Edition, why not then restore just the damaged page using database and log backups while users are online in the database? It could save you a lot of time and grief. I highly recommend you test this feature and understand exactly how it works before you rely on this possible strategy. An exercise at the end of this chapter helps you understand what is required to use this feature.

Back up system databases.

Don't forget the importance of your system databases. The failure to open these databases at startup can result in a failure to start SQL Server. If you can't restore from a backup, you could be looking at a longer recovery process to rebuild the system databases, attach user databases, re-create user logins, and so on. Because `tempdb` is re-created at each server startup, you (of course) don't back this up, but `model` is important because you may add objects in `model` for new databases, and `tempdb` requires `model` to be available to be created at server startup. I have an entire section of this chapter on recovery of system databases, and I think you will see that having a valid backup to restore makes your life easier.

Database and Transaction Log Best Practices

Don't delete your transaction log.

I hope one thing you get out of this chapter is how important your transaction log is to the logical consistency of your database. So, just because database recovery is taking a long time or the log is getting really big, don't think that simply shutting down SQL Server and deleting the log is a good idea. As with other scenarios, this is not uncommon for customers calling Microsoft PSS. They delete the log file and then call us wondering what to do because the database cannot be started.

If recovery is taking a long time, and you think it is stuck or will never recover, it is possible Microsoft PSS may have to help you rebuild the log, but consider contacting PSS first. You might just choose to restore from a backup, but if the backup is good, that is a much better alternative than just deleting your log.

The one scenario in which you can safely delete the log is if the database is detached and shut down cleanly and the recovery mode is `SIMPLE`. In this case, the log has nothing in it for recovery purposes, so you could delete it. Personally, I would use this method. I would simply shrink the log file using `DBCC SHRINKFILE`. Remember, if you choose to do this, you have to detach the database first.

Always detach before attaching a database.

The proper method to attach a database requires that you detach it first. Don't just shut down SQL Server, copy the database and log files to another computer, and then attach them. If you need to copy the database, use `BACKUP/RESTORE` or the Copy Database Wizard in Management Studio.

The process of detaching cleanly shuts down the database. Furthermore, if the database is suspect, detach fails in SQL Server 2005 (SQL Server 2000 didn't do this), preventing you from getting into a situation where you can't attach.

Moving the `resource` database.

If you need to move the resource database, there is a documented method to do this in SQL Server 2005 Books Online. However, you must move the database to the exact same drive and directory as the master database. Failure to do this will result in problems when trying to install a service pack or hotfix package.

Don't ignore runtime errors.

If you encounter any error related to a database that indicates possible consistency problems, don't ignore these or take care of them "when you have time." Take these seriously and put them on your high-priority list of things to do. The read retry feature is nice in

SQL Server 2005, but it means some underlying problem might exist with your disk system. A successful read retry one day could be a checksum failure the next. If you have any doubts or questions about these errors related to a possible database consistency problem and I don't have the information in this chapter, consult with experts such as MVPs in the SQL Server newsgroups, or contact Microsoft PSS.

DBCC CHECKDB Best Practices

Just use `DBCC CHECKDB`.

In SQL Server 2005, `DBCC CHECKDB` does it all. No need to run `CHECKALLOC` or `CHECKCATALOG` separately in SQL Server 2005. `CHECKDB` does it all. You may choose to use `CHECKTABLE` on individual tables, but if `CHECKDB` runs in a reasonable time frame, I recommend you check the entire database. You can also consider the `WITH PHYSICAL_ONLY` option to perform a reasonable minimum check for the database and reduce the time of overall execution.

Don't just run `REPAIR`.

I talk more about restoring a backup versus repair in the section on `CHECKDB` errors, but I'll say briefly here that I recommend that you not just blindly run `CHECKDB` with the `REPAIR_ALLOW_DATA_LOSS` option. Just because `DBCC CHECKDB` doesn't report errors after repair doesn't mean you don't have issues to deal with. Repair may de-allocate a page containing data rows, which now means you have logical inconsistency of your data. If you have to use repair because you can't restore a database, try to keep information about errors in `CHECKDB` and either a backup of the damaged database or a copy of the damaged page(s) before using repair. This is the only way to learn about the possible cause of the damage to the database.

Use `RESTORE` rather than emergency mode `REPAIR`.

Emergency mode repair is a great feature (especially for PSS). It makes the process to repair and rebuild a log simple and easy to use. But the key to its usage is the term *emergency*. The SQL Server development team put in this feature for situations that you simply didn't account for in your disaster recovery strategy. But don't rely on this feature. I hope you never have to use this in your usage and administration of SQL Server, but it is nice to know the feature exists should it be needed.

You can't overuse `CHECKDB`.

So how often should you use `CHECKDB`? Well, I would say first that you can't run this too often. It might affect performance because of its resource usage, but you won't burn up your hard drive by using this command every day. Having said that, you probably should use this command only on some type of regular, but minimal, basis. There is no single formula for how often to run this command, but here is my opinion of some checkpoints for when `CHECKDB` should be run:

- Whenever you see a critical error in the `ERRORLOG`, especially ones I've documented as runtime consistency problems.
- Before and after any SQL Server service pack or hotfix installations.

- Before and after any major database application upgrades.
- Before and after any Windows operating system service pack or hotfix installations or upgrade.
- Before and after any hardware or system maintenance such as drivers, firmware updates, new hardware installation, or replacement of hardware components.
- On a regular basis that makes you comfortable with the consistency of your database. (This may be daily, weekly, or monthly. I wouldn't go any longer than one month to use `CHECKDB` unless you have something like a read-only database you can easily re-create at any time.)

`CHECKDB` **on backup servers is not a guarantee.**

If you choose to restore backups to another server to keep an updated copy of your database available, be careful if regular `DBCC CHECKDB` on the backup server reports no errors. The primary server could have problems unique to that machine that are not carried into a restore on the backup server. Now if you restore a full database backup and immediately run `CHECKDB` on that restored backup, it is reasonable to assume the primary databases based on that backup are clean.

Data Recovery Troubleshooting Scenarios

I've organized the various type of data recovery scenarios I want to teach you how to troubleshoot into system database recovery, user database inaccessible, `BACKUP/RESTORE` failures, runtime consistency errors, and `CHECKDB` errors. Each section is self-contained so that you can focus on the area you are most interested in.

System Database Recovery

In most cases, recovering your database is your primary concern, because it contains the data that supports your application and business. However, understanding how to recover system databases can be important even though it occurs less often. This is because in most cases, a problem with a system database can mean SQL Server cannot start. The exception in this chapter is the MSDB database, but this too can be critical if you rely on it for its capability to run SQLAgent jobs.

Recovering Master

There are two possible scenarios where SQL Server cannot access any database including master or your user database:

- A resource problem opening database or log files.
- Recovery fails, causing the database to become `SUSPECT`.

A resource problem means that the engine encountered an error when trying to open a database or log file.

Scenario 1: Failure to Open the Master Database File (master.mdf)

Let's consider this scenario first. The master database is a unique database because it is the single database used to bootstrap the execution of the engine. Server-wide information is only stored in the master database, such as the registration of all other databases, login accounts, configuration values, error messages, and linked server information.

When the engine first starts, it must first open the master database to read the server-wide configuration information. At this point, if the master database file (`master.mdf`) cannot be opened, SQL Server does not start. The `ERRORLOG` would look something like this:

```
2006-01-28 11:07:37.06 Server   Microsoft SQL Server 2005 - 9.00.1399.06
(Intel X86)
     Oct 14 2005 00:33:37
     Copyright (c) 1988-2005 Microsoft Corporation
     Developer Edition on Windows NT 5.1 (Build 2600: Service Pack 2)
2006-01-28 11:07:37.06 Server   (c) 2005 Microsoft Corporation.
2006-01-28 11:07:37.06 Server   All rights reserved.
2006-01-28 11:07:37.06 Server   Server process ID is 4036.
2006-01-28 11:07:37.06 Server   Logging SQL Server messages in file 'C:\Program
Files\Microsoft SQL Server\MSSQL.1\MSSQL\LOG\ERRORLOG'.
2006-01-28 11:07:37.06 Server   This instance of SQL Server last reported using a
process ID of 2952 at 1/26/2006 5:04:21 PM (local) 1/26/2006 11:04:21 PM (UTC).
This is an informational message only; no user action is required.
2006-01-28 11:07:37.06 Server   Registry startup parameters:
2006-01-28 11:07:37.06 Server        -d C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\DATA\master.mdf2
2006-01-28 11:07:37.06 Server        -e C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\LOG\ERRORLOG
2006-01-28 11:07:37.06 Server        -l C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\DATA\mastlog.ldf
2006-01-28 11:07:37.07 Server   Error: 17113, Severity: 16, State: 1.
2006-01-28 11:07:37.07 Server   Error 2(The system cannot find the file
specified.) occurred while opening file 'C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\DATA\master.mdf2' to obtain configuration information at
startup. An invalid startup option might have caused the error. Verify your
startup options, and correct or remove them if necessary.
```

You can see that `Msg 17113` indicates a problem trying to read the configuration information from the master database file.

Troubleshooting Steps

Look at the error information right after the `17113` message. In this case, it says this:

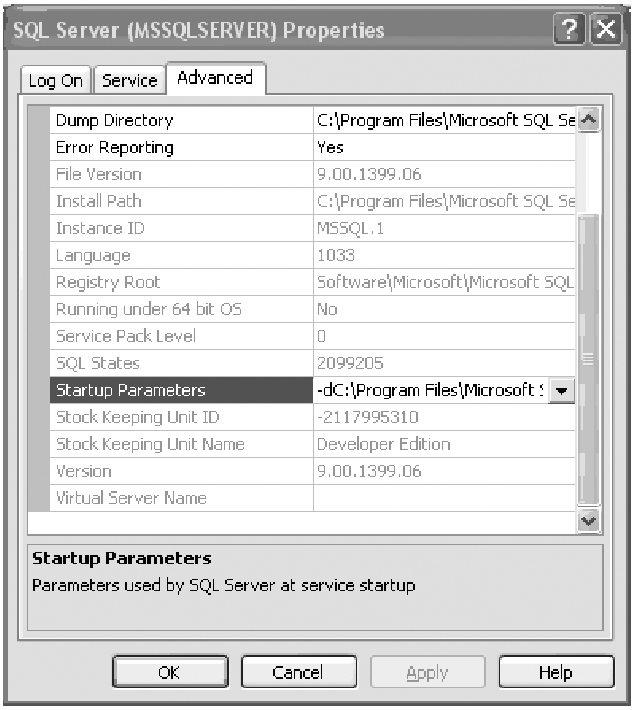
```
2006-01-28 11:07:37.07 Server   Error 2(The system cannot find the file specified.)
occurred while opening file 'C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\DATA\master.mdf2'
```

Error 2 is the Windows error returned from the Windows API `CreateFile` call. The text description follows the error number.

Determine the cause of the Windows error. In this case, error 2 means that the filename SQL Server passed to `CreateFile` does not exist. This means the path is wrong or the file does not exit. Check the full name of the path in the message. The standard name for the master database file is `master.mdf`, not `master.mdf2`.

If this is true, why does the engine think the name is `master.mdf2`? Because the master database is the bootstrap database, the engine has to get the location of the master database files from a source other than a database. It uses program parameters to do this. `-d` is used to specify the location of the master database file, and `-l` for the master transaction log file. Because SQL Server is normally run as a service, it must get the default startup parameter values from the registry.

Rather than edit the registry directly, use the SQL Server Configuration Manager. This is a good time to point out that any startup parameter setting or service configuration option (such as the startup account for SQL Server) should be done using the SQL Configuration Manager. Figure 2-3 shows what the interface looks like to change startup parameters for SQL Server.



Dump Directory	C:\Program Files\Microsoft SQL Se
Error Reporting	Yes
File Version	9.00.1399.06
Install Path	C:\Program Files\Microsoft SQL Se
Instance ID	MSSQL.1
Language	1033
Registry Root	Software\Microsoft\Microsoft SQL
Running under 64 bit OS	No
Service Pack Level	0
SQL States	2099205
Startup Parameters	-dC:\Program Files\Microsoft :
Stock Keeping Unit ID	-2117995310
Stock Keeping Unit Name	Developer Edition
Version	9.00.1399.06
Virtual Server Name	

Startup Parameters
Parameters used by SQL Server at service startup

FIGURE 2-3 Changing startup parameters for SQL Server

One tip when using this dialog box. The Startup Parameters is a single text field. To add a new parameter, you need to add it to the end of the parameters delimited by a semicolon.

Even though a failure to open the transaction log file can be resolved using the same technique, the errors reported in the ERRORLOG are slightly different:

```
2006-02-04 17:25:15.37 spid5s   Error: 17207, Severity: 16, State: 1.
2006-02-04 17:25:15.37 spid5s   FCB::Open: Operating system error 2(The system
cannot find the file specified.) occurred while creating or opening file
'C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\mastlog.ldf2'. Diagnose
and correct the operating system error, and retry the operation.
2006-02-04 17:25:15.40 spid5s   Error: 17204, Severity: 16, State: 1.
2006-02-04 17:25:15.40 spid5s   FCB::Open failed: Could not open file C:\Program
Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\mastlog.ldf2 for file number 2. OS
error: 2(The system cannot find the file specified.).
2006-02-04 17:25:15.48 spid5s   Error: 5120, Severity: 16, State: 101.
2006-02-04 17:25:15.48 spid5s   Unable to open the physical file "C:\Program
Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\mastlog.ldf2". Operating system
error 2: "2(The system cannot find the file specified.)".
```

Scenario 2: Master Database Recovery Failure

Like any user database, if recovery fails, the database is marked SUSPECT. However, one major difference is that the server shuts down when it detects that recovery has failed. Consider a situation where SQL Server attempts to start but fails. You look at the ERRORLOG and see the following:

```
2006-03-19 22:35:02.14 Server   Microsoft SQL Server 2005 - 9.00.1399.06
(Intel X86)
     Oct 14 2005 00:33:37
     Copyright (c) 1988-2005 Microsoft Corporation
     Developer Edition on Windows NT 5.1 (Build 2600: Service Pack 2)
2006-03-19 22:35:02.14 Server   (c) 2005 Microsoft Corporation.
2006-03-19 22:35:02.14 Server   All rights reserved.
2006-03-19 22:35:02.14 Server   Server process ID is 2040.
2006-03-19 22:35:02.14 Server   Logging SQL Server messages in file 'C:\Program
Files\Microsoft SQL Server\MSSQL.1\MSSQL\LOG\ERRORLOG'.
2006-03-19 22:35:02.15 Server   This instance of SQL Server last reported using a
process ID of 3628 at 3/19/2006 10:32:33 PM (local) 3/20/2006 4:32:33 AM (UTC).
This is an informational message only; no user action is required.
2006-03-19 22:35:02.15 Server   Registry startup parameters:
2006-03-19 22:35:02.15 Server        -d C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\DATA\master.mdf
2006-03-19 22:35:02.15 Server        -e C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\LOG\ERRORLOG
2006-03-19 22:35:02.15 Server        -l C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\DATA\mastlog.ldf
2006-03-19 22:35:02.18 Server   SQL Server is starting at normal priority base
(=7). This is an informational message only. No user action is required.
2006-03-19 22:35:02.18 Server   Detected 1 CPUs. This is an informational message;
no user action is required.
2006-03-19 22:35:02.56 Server   Using dynamic lock allocation. Initial allocation
of 2500 Lock blocks and 5000 Lock Owner blocks per node. This is an informational
message only. No user action is required.
2006-03-19 22:35:02.56 Server   Attempting to initialize Microsoft Distributed
```

```
Transaction Coordinator (MS DTC). This is an informational message only. No user
action is required.
2006-03-19 22:35:02.59 Server   The Microsoft Distributed Transaction Coordinator
(MS DTC) service could not be contacted. If you would like distributed transaction
functionality, please start this service.
2006-03-19 22:35:02.59 Server   Database Mirroring Transport is disabled in the
endpoint configuration.
2006-03-19 22:35:02.59 spid5s   Starting up database 'master'.
2006-03-19 22:35:04.34 spid5s   Error: 824, Severity: 24, State: 2.
2006-03-19 22:35:04.34 spid5s   SQL Server detected a logical consistency-based I/O
error: incorrect pageid (expected 1:506; actual 39321:-1717986919). It occurred
during a read of page (1:506) in database ID 1 at offset 0x000000003f4000 in file
'C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\master.mdf'. Additional
messages in the SQL Server error log or system event log may provide more detail.
This is a severe error condition that threatens database integrity and must be
corrected immediately. Complete a full database consistency check (DBCC CHECKDB).
This error can be caused by many factors; for more information, see SQL Server
Books Online.
2006-03-19 22:35:04.34 spid6s   Error: 922, Severity: 14, State: 1.
2006-03-19 22:35:04.34 spid6s   Database 'master' is being recovered. Waiting until
recovery is finished.
2006-03-19 22:35:04.34 spid6s   System Task System Task produced an error that was
not handled. Major: 9, Minor: 22, Severity:14, State:1
2006-03-19 22:35:04.34 spid5s   Error: 3313, Severity: 21, State: 2.
2006-03-19 22:35:04.34 spid5s   During redoing of a logged operation in database
'master', an error occurred at log record ID (2017:314:13). Typically, the specific
failure is previously logged as an error in the Windows Event Log service. Restore
the database from a full backup, or repair the database.
2006-03-19 22:35:04.34 spid5s   Cannot recover the master database. SQL Server
is unable to run. Restore master from a full backup, repair it, or rebuild
it. For more information about how to rebuild the master database, see SQL
Server Books Online.
```

Troubleshooting Steps

First, read the errors from the bottom up. You can see that the problem is a recovery problem for master, the error was during the redo of a log operation, and that the cause of that problem is an 824 error on the database page. The 824 error in this case is due to an incorrect page ID on page 1:506.

Because deferred transactions don't apply to the master database, your choices are the following:

- Restore a backup of the master database.
- Rebuild it using the setup program that comes with the installation medium.

Let's address how to restore a backup in this situation. You always have to start SQL Server in single user mode (using the `-m` startup parameter) to restore master, but in this case, SQL Server cannot be started because of the master recovery failure. Well, a nice trace flag helps you in this situation, trace flag 3607. Trace flag 3607 tells the engine to open just the master database, but don't run recovery on it. It is only to be used in this type of emergency situation, when master database recovery fails.

So the steps are as follows:

1. Start SQL Server with `-m` and `-T3607` as startup parameters using the SQL Server Configuration Manager.
2. Restore your backup of the master database.

   You see a message indicating SQL Server is being shut down:

   ```
   Processed 360 pages for database 'master', file 'master' on file 1.
   Processed 2 pages for database 'master', file 'mastlog' on file 1.
   The master database has been successfully restored. Shutting down SQL Server.
   SQL Server is terminating this process.
   ```

   SQL Server shuts down by design whenever you restore master. It is important to pay attention to the message to the client, because the `ERRORLOG` will not show a reason why SQL Server was shut down in this situation.
3. Remove the `-m` and `-T3607` startup parameters and restart SQL Server.

So if you don't happen to have a backup of the master database, your only choice is to rebuild it. After you rebuild it, you lose any previous information in master. Unfortunately, trace flag 3607 is only helpful to restore master. When you use this trace flag, any attempt to access an object in master results in the following error:

```
Msg 904, Level 16, State 1, Line 1
Database 32767 cannot be autostarted during server shutdown or startup.
```

Putting master in emergency mode won't help either, because it is not allowed for the master database.

So now that you are faced with the prospect of rebuiding master, how do you go about it? The first step is to find the installation media or source for your install (perhaps a network drive). Why? Because the method to rebuild the master database is contained within the SQL Server Setup program that only comes with the installation media. The steps to use setup to rebuild master are actually well documented in SQL Server Books Online in the section "Rebuilding System Databases, Rebuilding the Registry," so I don't list them here. An important point for you to keep in mind is that after rebuilding master, you must reapply any service packs or hotfixes you have previously installed, because they might have updated catalog information or the `resource` database.

The third scenario that I didn't call out specifically is simply a need to restore master because of a corruption problem or perhaps a change made that you want undone (for example, dropping some important logins). If the master database is fully available and `ONLINE`, all you need to do is `RESTORE` your backup.

Recovering Model

Just like the master database, if the model database cannot be started, SQL Server will not start successfully. If the problem was due to a problem with opening the model database file, the `ERRORLOG` shows entries such as the following:

```
2006-03-19 23:52:11.06 spid9s    Error: 17207, Severity: 16, State: 1.
2006-03-19 23:52:11.06 spid9s   FCB::Open: Operating system error 2(The system
cannot find the file specified.) occurred while creating or opening file
'C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\model.mdf'. Diagnose and
correct the operating system error, and retry the operation.
2006-03-19 23:52:11.06 Server   Server is listening on [ 127.0.0.1 <ipv4> 1434].
2006-03-19 23:52:11.06 spid9s   Error: 17204, Severity: 16, State: 1.
2006-03-19 23:52:11.06 spid9s   FCB::Open failed: Could not open file C:\Program
Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\model.mdf for file number 1. OS
error: 2(The system cannot find the file specified.).
2006-03-19 23:52:11.06 Server   Dedicated admin connection support was established
for listening locally on port 1434.
2006-03-19 23:52:11.06 spid9s   Error: 5120, Severity: 16, State: 101.
2006-03-19 23:52:11.06 spid9s   Unable to open the physical file "C:\Program
Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\model.mdf". Operating system error 2:
"2(The system cannot find the file specified.)".
2006-03-19 23:52:11.12 spid9s   Error: 945, Severity: 14, State: 2.
2006-03-19 23:52:11.12 spid9s   Database 'model' cannot be opened due to
inaccessible files or insufficient memory or disk space. See the SQL Server
errorlog for details.
2006-03-19 23:52:11.14 spid9s    Could not create tempdb. You may not have
enough disk space available. Free additional disk space by deleting other
files on the tempdb drive and then restart SQL Server. Check for additional
errors in the event log that may indicate why the tempdb files could not be
initialized.
```

This is similar to the situation with the master database; just the error numbers have changed. In this case, error 5120 contains the Windows error to explain why the model database file could not be opened. In the preceding scenario, it is Windows error 2, which means the file could not be found. Troubleshoot the same way as master, except the location for where the engine believes the file should exist is stored in the master database rather than in the registry. If I encountered this error, I would check to see whether the file existed in the path. If it did not, I would see if the `model.mdf` file existed anywhere on the drive. If I found it, I could copy it to the path as listed in the message or change the location where the engine should look for the file.

How do I do this if the engine will not start? Here are the steps:

1. Add the startup parameter `-T3608` to start SQL Server. This tells the engine to not start any database except master.
2. Use `ALTER DATABASE` and the `MODIFY FILE` option to change the path of the model database file to its correct path.
3. Shut down and restart SQL Server without `-T3608` so that all system and user databases can recover.

A more obscure scenario would be for recovery to fail for the model database. I say this because the model database is not changed very often by customers. Customers who do want to add objects to the model database so that every new database will have them usually do this infrequently. But should recovery fail for the model database, you need to use trace flag 3608 again to start the server so that you can restore model from a backup. This was not possible in SQL Server 2000, because the engine would attempt to use `tempdb` during the `RESTORE` command. Because `tempdb` first requires model to be started, the `RESTORE` command would fail. Now that restriction is removed, and `RESTORE` should work with trace flag 3608. What about that cool feature I told you about called emergency mode repair with `CHECKDB`? Is that possible for model? Does it make sense? First of all, it doesn't make sense. You don't want to create new databases based on a possible logically inconsistent model database. But it really doesn't matter anyway, because `ALTER DATABASE SET EMERGENCY` is not allowed on the model database. (For that matter, it isn't allowed on any system database.)

Recovering MSDB

The `msdb` database is considered a system database even though it is not required to start SQL Server. One reason that you can consider it a system database is that if you rebuild the master database with setup, you are in effect rebuilding all system databases, including `model`, `MSDB`, and the `resource` database. Furthermore, MSDB (in case you were wondering, MSDB stands for Microsoft System Database) is used to store important information such as backup/restore history and SQL Server Agent jobs. In fact, the SQL Server Agent service will not start if the `MSDB` database is not available.

The scenarios for problems accessing `MSDB` are the same as master and model. Either you can encounter a resource problem (such as problems opening the database/log files), or recovery could fail, causing the database to become `SUSPECT`.

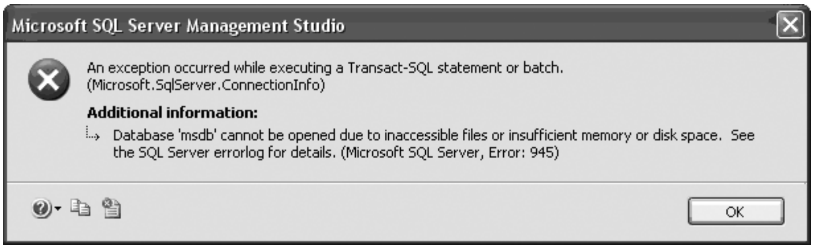
Because `MSDB` is more like a user database than `model`, you can use more-conventional techniques to recover it. For example, because SQL Server can start if a problem exists with `MSDB`, you don't need any special trace flags to restore a backup of `MSDB`. If the `MSDB` database becomes `SUSPECT`, you can just restore from a backup. If you don't have a backup available, you must rebuild the system databases using the same technique as you would rebuild `master` or `model`. In previous versions, you could get away with manually running the `instmsdb.sql` script in the `INSTALL` directory of SQL Server. Unfortunately, this technique is no longer that simple (the gyrations to make it work are pretty complex, and I don't think they are reliable), so the safest bet is to rebuild system databases.

If the `MSDB` database or log files cannot be opened, the database state appears as `RECOVERY_PENDING`, as described in the "Storage Internals" section. The errors you will see in the `ERRORLOG` look much like the scenario with model:

```
2006-03-20 18:58:47.71 spid12s   Error: 17207, Severity: 16, State: 1.
2006-03-20 18:58:47.71 spid12s   FCB::Open: Operating system error 2(The system
cannot find the file specified.) occurred while creating or opening file
'c:\program files\microsoft sql server\mssql.1\mssql\data\msdbdata.mdf'. Diagnose
and correct the operating system error, and retry the operation.
```

```
2006-03-20 18:58:47.71 spid12s   Error: 17204, Severity: 16, State: 1.
2006-03-20 18:58:47.71 spid12s   FCB::Open failed: Could not open file c:\program
files\microsoft sql server\mssql.1\mssql\data\msdbdata.mdf for file number 1. OS
error: 2(The system cannot find the file specified.).
2006-03-20 18:58:47.73 spid12s   Error: 5120, Severity: 16, State: 101.
2006-03-20 18:58:47.73 spid12s   Unable to open the physical file "c:\program
files\microsoft sql server\mssql.1\mssql\data\msdbdata.mdf". Operating system
error 2: "2(The system cannot find the file specified.)".
```

In fact, as you will see in the next section on user databases, these errors look the same as if a user database files could not be opened. The difference between this scenario and the model database is that the server is still up and running. You can then try to resolve the problem (perhaps by getting the `msdbdata.mdf` file in its right location) and simply execute `ALTER DATABASE msdb SET ONLINE` to start it up. By the way, don't get nervous when doing this in SQL Server Management Studio. If you attempt to connect to Object Explorer for the server in this state, you will get the error shown in Figure 2-4.


Microsoft SQL Server Management Studio

An exception occurred while executing a Transact-SQL statement or batch. (Microsoft.SqlServer.ConnectionInfo)

Additional information:

Database 'msdb' cannot be opened due to inaccessible files or insufficient memory or disk space. See the SQL Server errorlog for details. (Microsoft SQL Server, Error: 945)

FIGURE 2-4 Error message from attempting to connect to Object Explorer for the server

After you click OK, you won't see anything under the server name from Object Explorer. This is because Object Explorer tries to execute a stored procedure in the `msdb` database to show properties for SQL Server Agent after it connects. If this fails, it just doesn't show you anything. That's okay. Get `MSDB` back online, and then right-click the server instance listed. Select Disconnect, which removes the server instance from the screen. Now you can select the Connect drop-down option and reconnect to Object Explorer.

Recovering the Resource Database

You recall the resource database is read-only and "hidden." This means you can't reference this in commands such as `ALTER DATABASE`. So the techniques to recover from problems with it are different than even other system databases. For example, there is no way to `BACKUP` the resource database online. But because you can't modify it, there is no need to. The only way to back up the resource database is to back up the files themselves.

This scenario is very much the `model` database, because the server cannot start if it cannot open the `resource` database. In fact, the errors are identical to what you see in this situation for `model`. So look at the Windows error and resolve this to restart SQL Server. There is one big difference from the `model` situation. Even if you want to start SQL Server with `-T3608` to avoid starting the `resource` database, you really cannot do anything. What I mean is that you can't query any catalog view, because they require the resource database.

Furthermore, as I've said, T-SQL commands can't reference the `resource` database by name. Given this information, the only way to encounter a problem opening the resource database files is something occurring external to SQL Server (for example, someone going into the `DATA` directory and renaming the `mssqlsystemresource.mdf` file).

The other scenario could simply be damage to the resource database caused by some external factor (perhaps the portion of the disk containing this database was damaged). Because there is no way to `BACKUP` the database or `RESTORE` it, the only method to recover is to copy the resource database files themselves from the proper source. What is the proper source? This is where you have to manage these files a bit. For the RTM version of SQL Server 2005, no problem. These files are on the installation media. You may not have this lying around, which is why I recommend you copy them to some backup source you can get to easily. For any hotfixes or service packs you apply, you should also keep this copy of the files handy to replace.

When you start applying hotfixes or future service packs, you may ask, "How do I know if I've got the correct version to copy?" There is a method to find out the version of the resource database. Remember I told you to think of the resource database as DLL. The problem with versions is that the resource database files are not DLLs, so there is no way with Windows to put in a version value in the property of these files. So, the SQL Server development team built in the version to the `resource` database and provided you a way to query it:

```
select ServerProperty('ResourceVersion')
```

For the RTM version of SQL Server, this returns the following result:

```
------------------------------------------
9.00.1399
```

So how do I know whether this is the "right" version of the resource database? This version matches the engine's build number. And the query to find out the build number is the following:

```
select ServerProperty('ProductVersion')
```

For the original shipping version of SQL Server 2005 (known internally as the RTM version), this returns the following:

```
------------------------------------------
9.00.1399.06
```

You can ignore the .06 on this version for comparison's sake. This is just an internal number used as part of the final builds for SQL Server 2005 RTM. So if you run these two queries and you receive the same results for the first three numbers separated by the decimal, you have the correct resource database. If they don't match, you have a problem and need to find the correct version. This is why I recommend whenever you apply a hotfix or service pack, you take the resource database files copied to the `DATA` directory and keep them tucked away in a safe place. What would happen if these two versions differ? For SQL Server 2005 RTM, maybe nothing. The true answer is the behavior of catalog views and system objects in the engine is unpredictable. For future versions of the engine (hint: perhaps in the first service pack of SQL Server 2005), an error may be produced in the `ERRORLOG`, and the engine might not start.

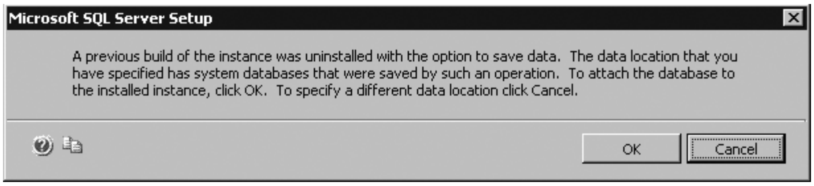
Failure to Create *tempdb*

As with previous versions of SQL Server, `tempdb` is created from scratch each time SQL Server is started. This means recovery is not run for `tempdb`. This does not mean there won't be problems creating `tempdb` at server startup. One such problem I've seen as reported by customers is not having enough disk space to create the `tempdb` files. By default, `tempdb` is created with a single 8MB database file and 512KB transaction log file. So no matter how large `tempdb` grows, when SQL Server is restarted, it is re-created with these default sizes. However, you can change these default sizes using `ALTER DATABASE` or SQL Server Management Studio. In fact, for performance reasons, I highly recommend you not rely on autogrowth of these files and change `tempdb` to a size that meets the needs of your application. (Unfortunately, I do not get into a recommendation here, but there is a section in Books Online called "Capacity Planning for tempdb.")

Reinstalling the Operating System

One other possible scenario to mention about system databases is reusing them if you have to reinstall the operating system. Consider the scenario where for whatever reason you decide to reinstall the Windows operating system. This will also require you to reinstall SQL Server. But if the drive containing the system database files and your user database files is valid and intact, why should you have to rebuild the system databases? In SQL Server 2000, this was exactly what you had to do. In SQL Server 2005, the setup program is smart enough to detect existing system databases.

When you run the setup for SQL Server, you can specify a path for the `DATA` directory where system databases will reside. Simply point that `DATA` directory to the same path where the valid system databases exist. If you do this, you see the dialog box from setup, as shown in Figure 2-5.



A previous build of the instance was uninstalled with the option to save data. The data location that you have specified has system databases that were saved by such an operation. To attach the database to the installed instance, click OK. To specify a different data location click Cancel.

FIGURE 2-5 Dialog box from setup

Just click OK and setup will use your existing system databases. When setup is done, you should be at the same state as you were before reinstalling the operating system (unless you had applied hotfixes or service packs, because they must be reapplied after this).

User Database Inaccessible

Let's shift focus now and talk about scenarios where you may have problems accessing your database. The scenarios are similar to some of the system database situations, but other options are available for recovery, because SQL Server can start even if your database cannot.

Database Marked *RECOVERY_PENDING*

Remember the database state diagram I explained earlier in the chapter. If there is any problem opening the database or transaction log files for a database, the database state is changed to `RECOVERY_PENDING`. In this scenario, if you attempt to access the database, you get the following error:

```
use pubs
go
Msg 945, Level 14, State 2, Line 1
Database 'pubs' cannot be opened due to inaccessible files or insufficient
memory or disk space. See the SQL Server errorlog for details.
```

Much like the case with `MSDB`, the `ERRORLOG` shows errors like these:

```
2006-03-23 14:26:05.99 spid18s   Error: 17207, Severity: 16, State: 1.
2006-03-23 14:26:05.99 spid18s   FCB::Open: Operating system error 2(The system
cannot find the file specified.) occurred while creating or opening file
'C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\pubs.mdf'. Diagnose and
correct the operating system error, and retry the operation.
2006-03-23 14:26:06.10 spid18s   Error: 17204, Severity: 16, State: 1.
2006-03-23 14:26:06.10 spid18s   FCB::Open failed: Could not open file C:\Program
Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\pubs.mdf for file number 1. OS error:
2(The system cannot find the file specified.).
2006-03-23 14:26:06.21 spid18s   Error: 5120, Severity: 16, State: 101.
2006-03-23 14:26:06.21 spid18s   Unable to open the physical file "C:\Program
Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\pubs.mdf". Operating system
error 2: "2(The system cannot find the file specified.)".
```

Unfortunately in this case, Management Studio does not give you any clue in Object Explorer that the database may have a problem (see Figure 2-6).

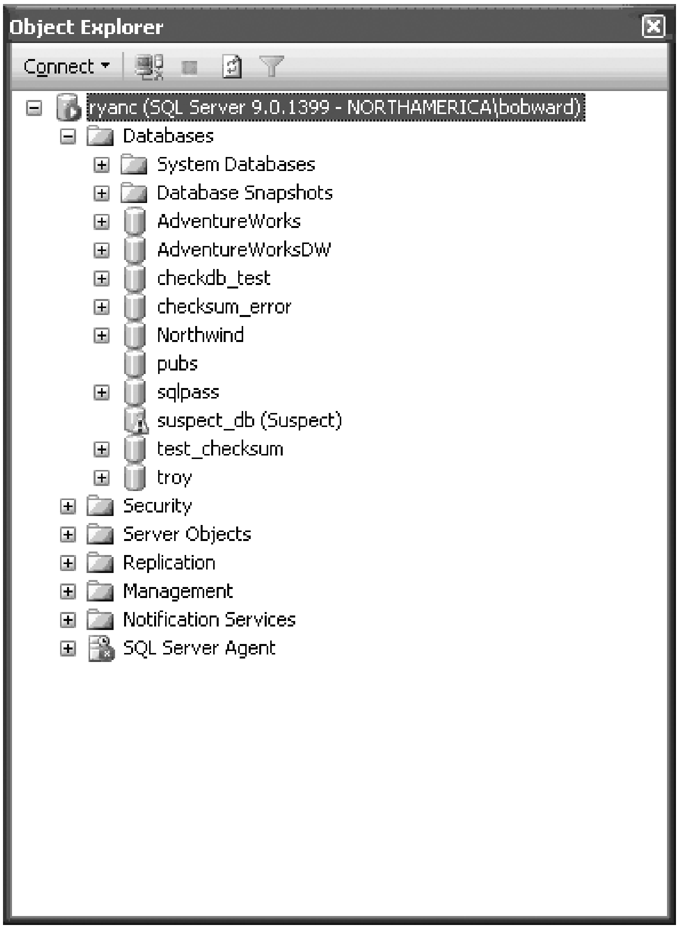
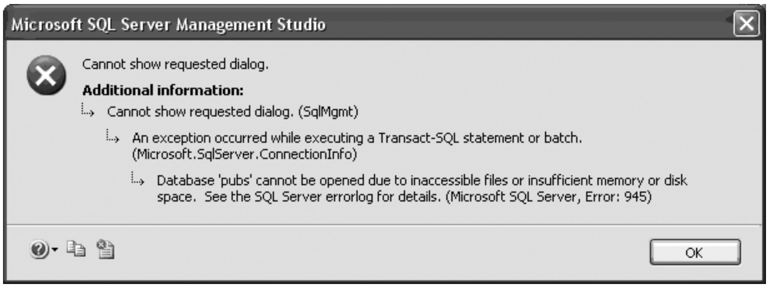


FIGURE 2-6 Management Studio does not indicate that the database may have a problem.

You can see, though, that next to the `pubs` database icon there is no plus sign (+) to examine objects in that database. This is a clue that there may be an issue with `pubs`. If you run into this situation, try to right-click the database icon and select Properties. This forces Management Studio to try and access the database (trying to obtain metadata about the state of files in the database).

In this particular scenario, you get a dialog box with any errors encountered trying to access properties for the database (in this case, `Msg 945`), as shown in Figure 2-7.


Microsoft SQL Server Management Studio

Cannot show requested dialog.

Additional information:

Cannot show requested dialog. (SqlMgmt)

An exception occurred while executing a Transact-SQL statement or batch. (Microsoft.SqlServer.ConnectionInfo)

Database 'pubs' cannot be opened due to inaccessible files or insufficient memory or disk space. See the SQL Server errorlog for details. (Microsoft SQL Server, Error: 945)

FIGURE 2-7 Dialog box with errors encountered while trying to access properties for the database

If you query the `sys.databases` catalog view, you can see the `pubs` database status of `RECOVERY_PENDING`:

```
select state_desc, name from sys.databases where name = 'pubs'
go

state_desc                                   name
------------------------------------------------------------ ----------------------
-----
RECOVERY_PENDING                                     pubs
```

If the transaction log file is not available, the database state is marked `RECOVERY_PENDING`, but errors reported in the `ERRORLOG` are slightly different:

```
2006-03-23 14:47:12.81 spid13s   Error: 17207, Severity: 16, State: 1.
2006-03-23 14:47:12.81 spid13s   FileMgr::StartLogFiles: Operating system error
2(The system cannot find the file specified.) occurred while creating or opening
file 'C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\pubs_log.LDF'.
Diagnose and correct the operating system error, and retry the operation.
2006-03-23 14:47:12.81 spid13s   File activation failure. The physical file
name "C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\pubs_log.LDF"
may be incorrect.
```

There is an interesting exception to this scenario when the transaction log file cannot be opened and the database uses a recovery mode of `SIMPLE`. If the database was shut down "cleanly," which means no recovery is required for the database on startup and the database recovery mode is `SIMPLE`, it is a safe operation for SQL Server to rebuild the transaction log. Why?

- A `SIMPLE` recovery has no log backups, so media recovery is not an issue.
- If there is no recovery to run on the database, no consistency is lost by rebuilding the log.

In this situation, on database startup, you will see some failures in the ERRORLOG but also a message indicating the transaction log is being rebuilt:

```
2006-03-23 14:42:59.28 spid13s   Error: 17207, Severity: 16, State: 1.
2006-03-23 14:42:59.28 spid13s   FileMgr::StartLogFiles: Operating system error
2(The system cannot find the file specified.) occurred while creating or opening
file 'C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\pubs_log.LDF'.
Diagnose and correct the operating system error, and retry the operation.
2006-03-23 14:42:59.36 spid13s   File activation failure. The physical file name
"C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\pubs_log.LDF" may be
incorrect.
2006-03-23 14:42:59.68 spid12s   Starting up database 'troy'.
2006-03-23 14:43:00.23 spid15s   CHECKDB for database 'test_checksum' finished
without errors on 2005-09-22 11:31:57.547 (local time). This is an informational
message only; no user action is required.
2006-03-23 14:43:01.06 spid12s   CHECKDB for database 'troy' finished without
errors on 2006-03-19 19:36:09.937 (local time). This is an informational message
only; no user action is required.
2006-03-23 14:43:01.17 spid13s   New log file 'C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\DATA\pubs_log.LDF' was created.
2006-03-23 14:43:01.17 spid13s   New log file 'C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\DATA\pubs_log.LDF' was created.
```

As I discuss later in this section, this comes in handy for attaching SIMPLE model databases with a single transaction log file. You don't even need the transaction log file to attach, because it is perfectly safe for SQL Server to rebuild one for you. Unfortunately, the log is rebuilt to a default size of 1MB, so you need to resize this to your needs.

Troubleshooting Steps

If the cause for RECOVERY_PENDING is inability to open the database files, the steps are simple, exactly as they were for MSDB:

1. Correct the problem that caused the file open failure.
2. Run ALTER DATABASE <dbname> SET ONLINE.
3. If ALTER DATABASE doesn't report any errors, you know that the database started successfully. You can also see your actions allowed the database to start in the ERRORLOG:

```
2006-03-23 15:07:44.18 spid51   Setting database option ONLINE to ON for
database pubs.
2006-03-23 15:07:44.25 spid51   Starting up database 'pubs'.
```

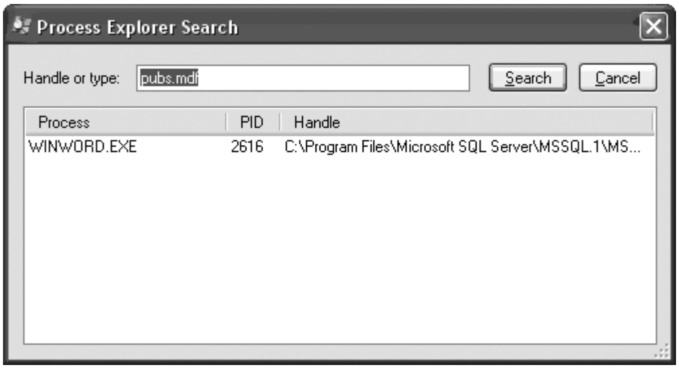
If no errors exist in the ERRORLOG after the "Starting up" message, the database should now be ONLINE (which you can also confirm by looking at sys.databases).

There is an interesting situation involving a failure to open a database or log file I want you to know about because it is tricky to troubleshoot. Look at the following ERRORLOG entries and notice the OS Error description for why the file cannot be opened:

```
2006-03-23 15:15:25.71 spid15s   Error: 17207, Severity: 16, State: 1.
2006-03-23 15:15:25.71 spid15s   FCB::Open: Operating system error 32(The process
cannot access the file because it is being used by another process.) occurred while
creating or opening file 'C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\DATA\pubs.mdf'. Diagnose and correct the operating system
error, and retry the operation.
2006-03-23 15:15:25.71 spid15s   Error: 17204, Severity: 16, State: 1.
2006-03-23 15:15:25.71 spid15s   FCB::Open failed: Could not open file C:\Program
Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\pubs.mdf for file number 1. OS error:
32(The process cannot access the file because it is being used by another
process.).
2006-03-23 15:15:25.71 spid15s   Error: 5120, Severity: 16, State: 101.
2006-03-23 15:15:25.71 spid15s   Unable to open the physical file "C:\Program
Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\pubs.mdf". Operating system
error 32: "32(The process cannot access the file because it is being used by
another process.)".
```

How can this situation occur? Why would SQL Server not be able to open a file because another program had it open? How do I find what program is doing this? There is a nice utility on the web called Process Explorer (which you can download from www.sysinternals.com).

To find which program has this open, launch Process Explorer and select the Find, Find Handle menu option. Type in the name of the file in the SQL error, and the program will tell you exactly what process has your database file handle open (see Figure 2-8).



Handle or type: pubs.mdf

Process	PID	Handle
WINWORD.EXE	2616	C:\Program Files\Microsoft SQL Server\MSSQL.1\MS...

FIGURE 2-8 What process your database file handle opened

In this case, someone has accidentally opened the file using Microsoft Word. Well, okay, it was me deliberately doing this to demonstrate the problem.

The same troubleshooting steps I've provided for database files apply to transaction log files. But what if you cannot resolve the problem with accessing the transaction log file? Do you have any other options? You do, but there are limits on how successful they will be.

In this situation, you can change the database state to EMERGENCY using ALTER DATABASE <dbname> SET EMERGENCY. At this point, you can access the database to query your data. But only SELECT statements to read data. You will encounter the following error if you try to modify the database:

```
Msg 3908, Level 16, State 1, Line 1
Could not run BEGIN TRANSACTION in database 'pubs' because the database is in
bypass recovery mode.
```

You can run some diagnostics like DBCC CHECKDB. However, you may get errors from this command, because the transaction log file was not available and recovery could not run. Recovery may be necessary to bring the database into a consistent state. So, your options are to copy all the data from your database to a new one or to use emergency mode repair. I covered this in the earlier section on enhancements to DBCC CHECKDB for SQL Server 2005.

Here are the steps to use CHECKDB in this situation (assuming the preceding example, where the pubs transaction log file could not be opened):

1. ALTER DATABASE pubs SET EMERGENCY.
2. ALTER DATABASE pubs SET SINGLE_USER.
3. DBCC CHECKDB (pubs, REPAIR_ALLOW_DATA_LOSS).
4. CHECKDB returns the following messages:

```
File activation failure. The physical file name "C:\Program Files\Microsoft
SQL Server\MSSQL.1\MSSQL\DATA\pubs_log.LDF" may be incorrect.
Warning: The log for database 'pubs' has been rebuilt. Transactional consistency
has been lost. The RESTORE chain was broken, and the server no longer has context
on the previous log files, so you will need to know what they were. You should run
DBCC CHECKDB to validate physical consistency. The database has been put in dbo-
only mode. When you are ready to make the database available for use, you will need
to reset database options and delete any extra log files.
DBCC results for 'pubs'.
Service Broker Msg 9675, State 1: Message Types analyzed: 14.
Service Broker Msg 9676, State 1: Service Contracts analyzed: 6.
Service Broker Msg 9667, State 1: Services analyzed: 3.
Service Broker Msg 9668, State 1: Service Queues analyzed: 3.
Service Broker Msg 9669, State 1: Conversation Endpoints analyzed: 0.
Service Broker Msg 9674, State 1: Conversation Groups analyzed: 0.
Service Broker Msg 9670, State 1: Remote Service Bindings analyzed: 0.
DBCC results for 'sys.sysrowsetcolumns'.
There are 626 rows in 6 pages for object "sys.sysrowsetcolumns".
DBCC results for 'sys.sysrowsets'.
There are 97 rows in 1 pages for object "sys.sysrowsets".
DBCC results for 'sysallocunits'.
There are 110 rows in 2 pages for object "sysallocunits".
DBCC results for 'sys.sysfiles1'.
There are 2 rows in 1 pages for object "sys.sysfiles1".
DBCC results for 'sys.syshobtcolumns'.
```

```
There are 626 rows in 6 pages for object "sys.syshobtcolumns".
DBCC results for 'sys.syshobts'.
There are 97 rows in 1 pages for object "sys.syshobts".
DBCC results for 'sys.sysftinds'.
There are 0 rows in 0 pages for object "sys.sysftinds".
DBCC results for 'sys.sysserefs'.
There are 110 rows in 1 pages for object "sys.sysserefs".
DBCC results for 'sys.sysowners'.
There are 14 rows in 1 pages for object "sys.sysowners".
DBCC results for 'sys.sysprivs'.
There are 123 rows in 1 pages for object "sys.sysprivs".
DBCC results for 'sys.sysschobjs'.
There are 99 rows in 2 pages for object "sys.sysschobjs".
DBCC results for 'sys.syscolpars'.
There are 497 rows in 9 pages for object "sys.syscolpars".
DBCC results for 'sys.sysnsobjs'.
There are 1 rows in 1 pages for object "sys.sysnsobjs".
DBCC results for 'sys.syscerts'.
There are 0 rows in 0 pages for object "sys.syscerts".
DBCC results for 'sys.sysxprops'.
There are 0 rows in 0 pages for object "sys.sysxprops".
DBCC results for 'sys.sysscalartypes'.
There are 30 rows in 1 pages for object "sys.sysscalartypes".
DBCC results for 'sys.systypedsubobjs'.
There are 0 rows in 0 pages for object "sys.systypedsubobjs".
DBCC results for 'sys.sysidxstats'.
There are 127 rows in 2 pages for object "sys.sysidxstats".
DBCC results for 'sys.sysiscols'.
There are 244 rows in 1 pages for object "sys.sysiscols".
DBCC results for 'sys.sysbinobjs'.
There are 23 rows in 1 pages for object "sys.sysbinobjs".
DBCC results for 'sys.sysobjvalues'.
There are 147 rows in 22 pages for object "sys.sysobjvalues".
DBCC results for 'sys.sysclsobjs'.
There are 14 rows in 1 pages for object "sys.sysclsobjs".
DBCC results for 'sys.sysrowsetrefs'.
There are 0 rows in 0 pages for object "sys.sysrowsetrefs".
DBCC results for 'sys.sysremsvcbinds'.
There are 0 rows in 0 pages for object "sys.sysremsvcbinds".
DBCC results for 'sys.sysxmitqueue'.
There are 0 rows in 0 pages for object "sys.sysxmitqueue".
DBCC results for 'sys.sysrts'.
There are 1 rows in 1 pages for object "sys.sysrts".
DBCC results for 'sys.sysconvgroup'.
There are 0 rows in 0 pages for object "sys.sysconvgroup".
DBCC results for 'sys.sysdesend'.
There are 0 rows in 0 pages for object "sys.sysdesend".
DBCC results for 'sys.sysdercv'.
There are 0 rows in 0 pages for object "sys.sysdercv".
DBCC results for 'sys.syssingleobjrefs'.
There are 163 rows in 1 pages for object "sys.syssingleobjrefs".
DBCC results for 'sys.sysmultiobjrefs'.
There are 133 rows in 1 pages for object "sys.sysmultiobjrefs".
DBCC results for 'sys.sysdbfiles'.
There are 2 rows in 1 pages for object "sys.sysdbfiles".
```

```
DBCC results for 'sys.sysguidrefs'.
There are 0 rows in 0 pages for object "sys.sysguidrefs".
DBCC results for 'sys.sysqnames'.
There are 91 rows in 1 pages for object "sys.sysqnames".
DBCC results for 'sys.sysxmlcomponent'.
There are 93 rows in 1 pages for object "sys.sysxmlcomponent".
DBCC results for 'sys.sysxmlfacet'.
There are 97 rows in 1 pages for object "sys.sysxmlfacet".
DBCC results for 'sys.sysxmlplacement'.
There are 17 rows in 1 pages for object "sys.sysxmlplacement".
DBCC results for 'sys.sysobjkeycrypts'.
There are 0 rows in 0 pages for object "sys.sysobjkeycrypts".
DBCC results for 'sys.sysasymkeys'.
There are 0 rows in 0 pages for object "sys.sysasymkeys".
DBCC results for 'sys.syssqlguides'.
There are 0 rows in 0 pages for object "sys.syssqlguides".
DBCC results for 'sys.sysbinsubobjs'.
There are 0 rows in 0 pages for object "sys.sysbinsubobjs".
DBCC results for 'publishers'.
There are 8 rows in 1 pages for object "publishers".
DBCC results for 'titles'.
There are 18 rows in 1 pages for object "titles".
DBCC results for 'titleauthor'.
There are 25 rows in 1 pages for object "titleauthor".
DBCC results for 'stores'.
There are 6 rows in 1 pages for object "stores".
DBCC results for 'sales'.
There are 21 rows in 1 pages for object "sales".
DBCC results for 'roysched'.
There are 86 rows in 1 pages for object "roysched".
DBCC results for 'discounts'.
There are 3 rows in 1 pages for object "discounts".
DBCC results for 'jobs'.
There are 14 rows in 1 pages for object "jobs".
DBCC results for 'pub_info'.
There are 8 rows in 1 pages for object "pub_info".
DBCC results for 'employee'.
There are 43 rows in 1 pages for object "employee".
DBCC results for 'x'.
There are 0 rows in 0 pages for object "x".
DBCC results for 'sys.queue_messages_1977058079'.
There are 0 rows in 0 pages for object "sys.queue_messages_1977058079".
DBCC results for 'sys.queue_messages_2009058193'.
There are 0 rows in 0 pages for object "sys.queue_messages_2009058193".
DBCC results for 'sys.queue_messages_2041058307'.
There are 0 rows in 0 pages for object "sys.queue_messages_2041058307".
DBCC results for 'authors'.
There are 23 rows in 1 pages for object "authors".
CHECKDB found 0 allocation errors and 0 consistency errors in database 'pubs'.
DBCC execution completed. If DBCC printed error messages, contact your system
administrator.
```

Pay special attention to the following message in this chain:

```
Warning: The log for database 'pubs' has been rebuilt. Transactional consistency
has been lost. The RESTORE chain was broken, and the server no longer has context
on the previous log files, so you will need to know what they were. You should run
DBCC CHECKDB to validate physical consistency. The database has been put in dbo-
only mode. When you are ready to make the database available for use, you will need
to reset database options and delete any extra log files.
```

What does this message mean? Well, it has a bunch of warnings, so let me summarize them for you:

- Transactional consistency

  Even though `DBCC CHECKDB` reports no errors, it doesn't mean that the database is logically consistent. What I mean is that since the log was rebuilt there could have been modifications to the database that need to be redone or undone during recovery for transactions to be consistent.

  Consider the following scenario. Your application credits a customer account for $10,000. The rows on the database page are updated to reflect a credit to the customer's account. Let's say the user of the application is not done with this customer, so by design the transaction for this operation is still active. But, because of memory pressure in the engine, this page is flushed to disk. To ensure consistency, SQL Server also flushes the log records associated with the transaction to disk. But the user realizes he made a mistake and clicks the Cancel button. This causes the application to roll back the transaction. In memory, the page and log records are changed. Within the next few minutes, a serious problem occurs on the server, and it crashes and reboots. SQL Server is robust to crash recovery. Even though the database page on disk shows a credit for $10,000, the transaction log never shows this transaction was committed. So when recovery runs, the modification undone. But what if the engine encountered an error opening the transaction log file and you choose to use emergency mode repair? Recovery never ran, so a credit exists for $10,000 for this customer account. Who would be able to find this problem? Could this really occur? The timing might be difficult, but who would want to take this risk? This is why the SQL Server development team put the text in this message and is why Microsoft PSS always has a serious conversation with customers who want to rebuild their transaction log.

- `RESTORE` chain broken

  Because we rebuild the transaction log, your backup log chain sequence is broken. You cannot rely on these backups, and you need to take a full backup of the database to restart a new backup sequence.

- Database options

  The database is left in `SINGLE_USER` mode and is changed to the `SIMPLE` recovery model.

The SQL Server development team put emergency mode repair in the product for customers to use, but only, as the name implies, for emergency situations. The development team also put in this message so that it would be clear to you that the server cannot vouch for the transactional consistency of your database. Just to make sure you understand that you or another DBA performed this operation, the server writes the following messages to the `ERRORLOG`:

```
2006-03-23 15:37:48.01 spid51   Warning: The log for database 'pubs' has been
rebuilt. Transactional consistency has been lost. The RESTORE chain was
broken, and the server no longer has context on the previous log files, so
you will need to know what they were. You should run DBCC CHECKDB to validate
physical consistency. The database has been put in dbo-only mode. When you are
ready to make the database available for use, you will need to reset database
options and delete any extra log files.
2006-03-23 15:37:48.01 spid51   Warning: The log for database 'pubs' has been
rebuilt. Transactional consistency has been lost. The RESTORE chain was broken, and
the server no longer has context on the previous log files, so you will need to
know what they were. You should run DBCC CHECKDB to validate physical consistency.
The database has been put in dbo-only mode. When you are ready to make the database
available for use, you will need to reset database options and delete any extra log
files.
2006-03-23 15:37:48.57 spid51   EMERGENCY MODE DBCC CHECKDB (pubs,
repair_allow_data_loss) executed by NORTHAMERICA\bobward found 0 errors and
repaired 0 errors. Elapsed time: 0 hours 0 minutes 0 seconds.
```

Handling Deferred Transactions

As discussed in the "Storage Internals" section, when recovery fails for a user database but the cause of the recovery is damage to a database page, SQL Server may not mark the database `SUSPECT`. Instead, the engine marks a bit on the damaged page in the header (set to a status called `RestorePending`). If the transaction associated with the operation that encountered the damaged page is still active after redo, all locks are held that are part of the transaction, and undo for the transaction is skipped (or deferred). The good news is that the database is still `ONLINE` and not `SUSPECT`. Only users who need to access this page or access data associated with any deferred transaction are affected.

How do you know whether recovery had to defer transactions but not mark the database `SUSPECT`?

- You see evidence in the `ERRORLOG`.
- You encounter an error when accessing a page that is marked `RestorePending`.
- You are blocked by `session_id = -3`.

The choices and steps for recovering a deferred transaction are simple:

- Restore a backup. But in this case, because only a single page is damaged, you could use online page restore to keep the rest of the database online and affect only the deferred transaction.

- If you don't have a backup to use, you can repair the page (but lose the data on it) by using `DBCC CHECKDB` and the `REPAIR_ALLOW_DATA_LOSS` option.

The steps for using page-level restore are documented in Books Online in the section "Performing Page Restores." Although I won't add anything about these steps in this section, there is an exercise in the end of this chapter to walk you through using this feature during a data recovery scenario.

Finding the root cause for a deferred transaction is not complicated, because the cause is damage to the database page, as outlined in the `ERRORLOG` (for example, a checksum error on a page). Therefore, you must investigate the cause of this problem just like you would for any checksum failure. As I mentioned earlier in this chapter, the cause of a checksum page error is not SQL Server but a problem in the IO path.

Database Marked *SUSPECT*

Suppose recovery fails and one of the conditions for a deferred transaction doesn't qualify or the failure is not associated with a single database page (for example, damage to the transaction log is detected). In this situation, the engine changes the database state to `SUSPECT`.

The following is an example of an `ERRORLOG` entry for a database marked `SUSPECT`:

```
2006-03-26 22:14:10.98 spid13s   Error: 824, Severity: 24, State: 4.
2006-03-26 22:14:10.98 spid13s   SQL Server detected a logical consistency-based
I/O error: (bad checksum). It occurred during a read of page (0:-1) in database ID
8 at offset 0x00000000013800 in file 'C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\DATA\suspect_db_log.LDF'. Additional messages in the SQL
Server error log or system event log may provide more detail. This is a severe
error condition that threatens database integrity and must be corrected
immediately. Complete a full database consistency check (DBCC CHECKDB). This error
can be caused by many factors; for more information, see SQL Server Books Online.
2006-03-26 22:14:11.43 spid13s   Error: 3414, Severity: 21, State: 1.
2006-03-26 22:14:11.43 spid13s   An error occurred during recovery, preventing
the database 'suspect_db' (database ID 8) from restarting. Diagnose the
recovery errors and fix them, or restore from a known good backup. If errors
are not corrected or expected, contact Technical Support.
```

Any attempt to access the database results in the following error:

```
Msg 926, Level 14, State 1, Line 1
Database 'suspect_db' cannot be opened. It has been marked SUSPECT by
recovery. See the SQL Server errorlog for more information.
```

Fortunately, Management Studio gives us a visual clue to the status of the database (see Figure 2-9).

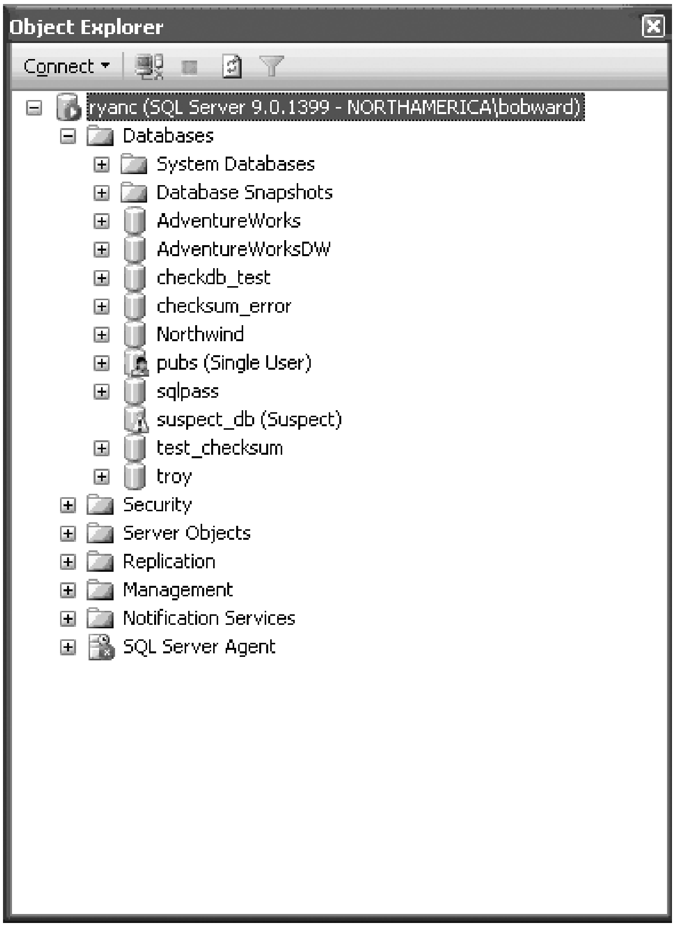


FIGURE 2-9 Management Studio gives a visual clue to the status of the database

As in the case of a deferred transaction, your options to recover are similar:

- You can restore from a backup. Page-level restore is not an option here, because the problem is not specific to a database page.
- Use `DBCC CHECKDB` for emergency mode repair. I outlined how this works when I discussed the scenario of the transaction log being unavailable.
- If emergency mode repair does not work, you could change the database state to `EMERGENCY` and try to copy as much data as possible.

Remember that the `SUSPECT` state for a database is temporary. This means to find the original cause for why the database was marked `SUSPECT`, you can either restart SQL Server or you can try to change the database state to `ONLINE`. Consider the example of the database `suspect_db`. If I try to change the database state to `ONLINE`, I get the following messages:

```
Msg 926, Level 14, State 1, Line 1
Database 'suspect_db' cannot be opened. It has been marked SUSPECT by recovery. See
the SQL Server errorlog for more information.
Msg 5069, Level 16, State 1, Line 1
ALTER DATABASE statement failed.
Msg 824, Level 24, State 4, Line 1
SQL Server detected a logical consistency-based I/O error: (bad checksum). It
occurred during a read of page (0:-1) in database ID 8 at offset 0x00000000013800
in file 'C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\DATA\suspect_db_log.LDF'. Additional messages in the SQL
Server error log or system event log may provide more detail. This is a severe
error condition that threatens database integrity and must be corrected
immediately. Complete a full database consistency check (DBCC CHECKDB). This error
can be caused by many factors; for more information, see SQL Server Books Online.
Msg 3414, Level 21, State 1, Line 1
An error occurred during recovery, preventing the database 'suspect_db'
(database ID 8) from restarting. Diagnose the recovery errors and fix them, or
restore from a known good backup. If errors are not corrected or expected,
contact Technical Support.
```

The first message after 5069 is the root cause of the `SUSPECT` failure, which in this example is a checksum failure (`Msg 824`) when reading the transaction log. (Yes, the transaction log uses checksum technology, too, to detect IO consistency.)

Attach Database Failures

It is possible to encounter different types of errors when attaching a database. First, let me eliminate one consideration that could occur in previous versions of SQL Server. You can't detach a database whose state is `SUSPECT`. Attempting to do this results in the following error:

```
Msg 3707, Level 16, State 2, Line 1
Cannot detach a suspect database. It must be repaired or dropped.
```

Although this may be true, you could technically do something you are not supposed to do. You could shut down SQL Server and still use the files from a `SUSPECT` database to attach to another server. In this case, the attach fails because the database was `SUSPECT`. Of course, you are supposed to *always* detach a database before you attach it. I've just now showed you an example of why it is important to do this.

Most problems I see with attaching databases have to do with missing some of the files associated with the database. If the database was detached properly (which means it was cleanly shut down) and the database has only one transaction log file, all you need to attach the database are the database files. The log will be rebuilt safely as I've described earlier when the log file is not available at startup (even for a database using `FULL` recovery). But if you do not have all the transaction log files as they existed for the original database, the server returns the following error:

```
File activation failure. The physical file name "C:\Program Files\Microsoft
SQL Server\MSSQL.1\MSSQL\DATA\test_attach_log2.ldf" may be incorrect.
The log was not rebuilt because there is more than one log file.
Msg 1813, Level 16, State 2, Line 1
Could not open new database 'test_attach'. CREATE DATABASE is aborted
```

In SQL Server 2000, you had no options at this point. SQL Server 2005 offers a new option to `FOR ATTACH_REBUILD_LOG`. This option still requires the database to be detached properly, but it lets you rebuild the log if you don't have all transaction log files.

BACKUP/RESTORE Failures

Because `BACKUP` and `RESTORE` are your most important tools to recover your data, I think it's important to briefly discuss some scenarios where these commands can fail and how to handle them.

BACKUP Failures

The T-SQL `BACKUP` command can fail for various reasons, some of them simpler to solve than others. For example, if you try to use `BACKUP LOG` and the database is using `SIMPLE` recovery, you will encounter the following error:

```
Msg 4208, Level 16, State 1, Line 1
The statement BACKUP LOG is not allowed while the recovery model is SIMPLE. Use
BACKUP DATABASE or change the recovery model using ALTER DATABASE.
Msg 3013, Level 16, State 1, Line 1
BACKUP LOG is terminating abnormally.
```

This error is descriptive and tells you exactly the problem. Other errors or problems are not as intuitive and simple to resolve, so I focus on those in this section.

Media Failures

One of the most common issues I've seen in technical support regarding backup is a failure writing to the target media. Target media is the destination for the backup stream, such as a disk file(s) or tape.

Tape failures were more common with previous releases of SQL Server because tape was the most common format for backing up large databases. I don't see as many of these cases anymore, but when they do occur, the cause of the failure is almost always a problem with the tape drive, the tape, or device drivers associated with the tape drive. I've seen this time and time again. Here is a sample `ERRORLOG` showing a failure writing to the tape that doesn't appear very intuitive.

```
spid324   BackupMedium::ReportIoError: write failure on backup device '\\.\tape0'.
Operating system error 1117(The request could not be performed because of an I/O
device error.).
```

The steps to resolve a problem such as this or other tape errors is to focus on the tape drive, tape, or device drivers. Try a different tape or a different tape drive. Make sure the device drivers for the tape drive are up-to-date. One common misconception for troubleshooting a problem with writing to tape is that if the Windows `NTBACKUP` program works, the problem must be SQL Server. Although it is a good idea to see whether you can back up a file using `NTBACKUP` to the tape drive, if it works but you still encounter an error with SQL Server, the problem could still be the tape system. This is because SQL Server uses Windows API calls that `NTBACKUP` doesn't when writing to the tape.

Before I talk about some common issues when writing to a disk drive, one simple troubleshooting step to help in many backup cases is to simplify what you are backing up. For example, if you are struggling to back up a 30GB database, try backing up just a small database, such as the sample AdventureWorks database. If you cannot back up even a small sample database, there is a good chance you should focus on the target media, whether it is a tape system, local disk, SAN, or network drive. If a small database sample works, but your database does not, it could still be the target media system, but there is a little more evidence that it could be a SQL Server problem. This is good information to give to a technical support engineer should you decide to contact Microsoft for assistance.

Let's talk about a few more common scenarios that you could encounter today—failures writing to a local disk, SAN, or network drive. As with tape, the cause of these problems is almost always something to do with the target media, current operating system environment, or target operating system environment. Let's take a look at two possible scenarios you may encounter where the solution to the problem is not obvious.

```
2005-05-02 09:15:28.05 spid69 BackupMedium::ReportIoError: write failure on backup
device 'G:\MSSQL\Backups\caamdb.BAK'. Operating system error 64(The specified
network name is no longer available.).
```

In this situation, the error is indicative of a network problem. The error has nothing to do with SQL Server. You should troubleshoot this problem as though there was a network I/O issue communicating with whatever disk is mapped to the G: drive. This may even be a SAN drive.

```
BackupMedium::ReportIoError: read failure on backup device
'\\FFGPS\D$\MSSQL\BACKUP\FSPRD_DB.BAK'. Operating system error
1450(Insufficient system resources exist to complete the requested service.).
```

In this situation, the error is coming from Windows when writing the backup file. I've especially seen this situation when backing up to a network drive. I've also seen this problem with the use of the `/3GB` switch in `boot.ini` because there is not enough nonpaged pool memory for Windows to handle the I/O associated with the backup. Your choices are 1) Don't back up to a networked drive, 2) Remove the use of `/3GB`, 3) Tune nonpaged pool according to the guidelines in Knowledge Base article 304101.

CHECKSUM Failures

I discussed earlier in the chapter a new feature for SQL Server 2005 called backup checksum. When the `WITH CHECKSUM` option is specified for the `BACKUP` command, the engine verifies the checksum for any pages it reads from the database file that previously had a checksum calculated for it. If any page fails this verification check, `BACKUP` fails with the following error:

```
Msg 3043, Level 16, State 1, Line 1
BACKUP 'test_checksum' detected an error on page (1:177) in file 'C:\Program
Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\test_checksum.mdf'.
Msg 3013, Level 16, State 1, Line 1
BACKUP DATABASE is terminating abnormally.
```

The `ERRORLOG` shows the following information:

```
2006-08-25 14:18:21.49 Backup      Error: 3043, Severity: 16, State: 1.
2006-08-25 14:18:21.49 Backup      BACKUP 'test_checksum' detected an error on page
(1:177) in file 'C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\DATA\test_checksum.mdf'.
2006-08-25 14:18:21.52 Backup      Error: 3041, Severity: 16, State: 1.
2006-08-25 14:18:21.52 Backup      BACKUP failed to complete the command
BACKUP DATABASE test_checksum. Check the backup application log for detailed
messages.
```

Your choices here are simple:

- Correct the checksum problem.
- Remember the cause of the problem is something within the disk system, so be sure to check all is well here first. I say this because if you choose to repair the database to correct the problem, but the disk still has problems, you will continue to run into other inconsistency issues. To repair the current problem, you can choose to restore from a backup (remember page-level backup restore is a possible option) or you can use `DBCC CHECKDB` to repair the page (which will result in data loss as the page will be de-allocated to "fix" it).
- Ignore the error and continue the `BACKUP`.

Let's say you just want to go ahead and back up the database ignoring this error, knowing it will be a bad page. You have an option with `BACKUP` called `CONTINUE_AFTER_ERROR` that will let you do just that. If you use this option, you will get the following warning after the `BACKUP` completes:

```
Processed 208 pages for database 'test_checksum', file 'test_checksum' on
file 1.
Processed 1 pages for database 'test_checksum', file 'test_checksum_log' on file 1.
BACKUP WITH CONTINUE_AFTER_ERROR successfully generated a backup of the damaged
database. Refer to the SQL Server error log for information about the errors that
were encountered.
BACKUP DATABASE successfully processed 209 pages in 1.879 seconds (0.910
MB/sec).
```

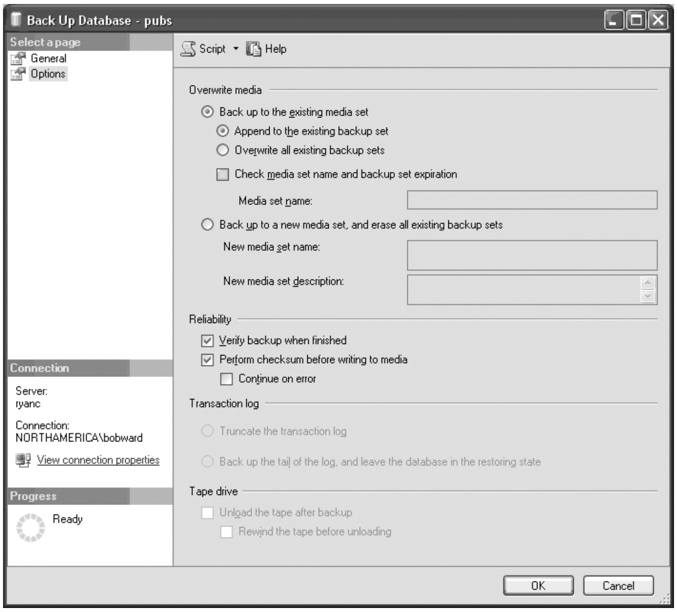
The ERRORLOG will contain the following information:

```
2006-08-25 14:20:30.83 Backup      Error: 3043, Severity: 16, State: 1.
2006-08-25 14:20:30.83 Backup      BACKUP 'test_checksum' detected an error on page
(1:177) in file 'C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\DATA\test_checksum.mdf'.
2006-08-25 14:20:31.03 Backup      Database backed up. Database:
test_checksum, creation date(time): 2006/02/22(11:35:35), pages dumped: 219,
first LSN: 22:16:26, last LSN: 22:28:1, number of dump devices: 1, device
information: (FILE=1, TYPE=DISK: {'c:\test_checksum.bak'}). This is an
informational message only. No user action is required.
```

Why would you consider this option? One reason may be that you want to back up the database in its current state before trying to repair it and correct the damaged page. I'm not saying repair will cause problems for you so you must back up your current database, but I've met careful DBAs who want to back up the current state of the database before trying any fairly major modification operation. The only way for you to do this is to use the CONTINUE_AFTER_ERROR option.

One point of clarification about the WITH CHECKSUM option for BACKUP. The BACKUP command itself does not verify any data after writing it to disk. In other words, BACKUP doesn't verify checksums, write the backup stream checksum, and then go back and verify everything. What it does is verify database page checksums as it reads pages from the database file, write these pages to the backup media, and then write a backup checksum value based on all data written to the media. You can do this yourself by immediately executing RESTORE WITH VERIFYONLY right after the BACKUP.

If you use Management Studio to back up a database, you are presented with an Options dialog box where you can select to use checksum and verify the integrity of the database (see Figure 2-10).



Back Up Database - pubs

Select a page: General, Options

Script Help

Overwrite media

[✓ Back up to the existing media set]

[✓ Append to the existing backup set]

Overwrite all existing backup sets

Check media set name and backup set expiration

Media set name:

Back up to a new media set, and erase all existing backup sets

New media set name:

New media set description:

Reliability

[✓ Verify backup when finished]

[✓ Perform checksum before writing to media]

Continue on error

Transaction log

Truncate the transaction log

Back up the tail of the log, and leave the database in the restoring state

Tape drive

Unload the tape after backup

Rewind the tape before unloading

Connection

Server:
ryanc

Connection:
NORTHAMERICA\bobward

View connection properties

Progress

Ready

OK Cancel

FIGURE 2-10 Option to use checksum and verify the integrity of the database

Look at the options in the Reliability section of the dialog box. If you select these options and use the Script menu at the top of the dialog box (this is a nice feature in Management Studio where you can generate scripts based on options picked in a dialog box for different types of tasks), the resulting T-SQL script will look like this:

```
BACKUP DATABASE [pubs] TO DISK = N'c:\pubs.bak' WITH NOFORMAT, NOINIT, NAME =
N'pubs-Full Database Backup', SKIP, NOREWIND, NOUNLOAD, STATS = 10, CHECKSUM
GO
declare @backupSetId as int
select @backupSetId = position from msdb..backupset where database_name=N'pubs' and
backup_set_id=(select max(backup_set_id) from msdb..backupset where
database_name=N'pubs' )
if @backupSetId is null begin raiserror(N'Verify failed. Backup information for
database ''pubs'' not found.', 16, 1) end
RESTORE VERIFYONLY FROM DISK = N'c:\pubs.bak' WITH FILE = @backupSetId, NOUNLOAD,
NOREWIND
GO
```

This is effectively what Management Studio (through SMO) will use to back up the database with checksum and then verify the backup when it is complete.

Performance Problems: Memory

One possible scenario you may encounter will not show up as an error message. The issue is performance problems with your `BACKUP`. Perhaps you have seen this before and were not sure as to the cause. We have had customers contact technical support and say that their backups are slowing down and in some cases taking twice as long. In some situations, this degradation has occurred over time. Let's explore how to recognize this behavior and what you can do to resolve it.

When customers have contacted us and complained about backup performance, we quickly noticed in most cases Performance Monitor would show a significant drop in backup throughput via the following counter: Physical Disk/Avg Disk Bytes Per Transfer. We observed this for the disk drive where the customer was writing backup. We noticed that this value on that drive was something much lower than we expected. For a normal SQL Server disk backup, this number should be close to 1MB. This is because we knew internally in SQL Server that the backup code will write out buffers of ~1MB in size to the target disk. The numbers we observed were more like 64KB. What this means is that the SQL engine is writing out a large number of small disk writes versus a smaller number of larger writes. This is not efficient for writing out to a disk drive (assuming no bottlenecks exist).

After some investigation, we discovered the problem. The SQL Server backup code calculates what will be its buffer size (called `MAXTRANSERSIZE`) based on the number of files for the database/log and the number of target disk drives. (Remember you can back up to multiple disk files at the same time, called a stripe dump.) The backup code then takes the total number of buffers and tries to allocate memory for them at a default size of 960KB. If any of these memory allocations fail, the code *downgrades* its `MAXTRANSFERSIZE` to 64KB. In SQL Server 2005 (and in SQL Server 2000 SP4), if you observe the following message in the `ERRORLOG`:

```
2005-12-02 02:00:21.93 spid100  Downgrading backup buffers from 960K to 64K
```

you know that the default `MAXTRANSERSIZE` could not work. As with any memory problem, the issue is not having enough free memory for these allocations. But here is the twist to this problem. The issue is not having enough contiguous memory to allocate a 960KB block. Furthermore, the memory for the backup buffers doesn't come from the SQL Server buffer pool. This comes from the remaining virtual address space after the buffer pool reserves its memory. This is a precious resource for systems with 2GB+ of physical memory, because the buffer pool consumes most of the virtual address space. This means there are two resolutions to this problem:

- Free up more virtual address space or find out why it is fragmented.
- Use the new `MAXTRANSFERSIZE` for a lower default setting than 960KB.

RESTORE Failures

Like `BACKUP`, there are various reasons for restore failing, but two that are the most complex are as follows:

- A consistency problem with the backup media
- A failure during recovery

The second scenario should be investigated just like the scenarios I've already described for deferred transactions and a SUSPECT database. This is because the problems are the same. One is a failure during crash recovery (when the database comes online), and the other is a failure of recovery during media recovery (when the database is being restored).

The first case is the one for us to focus on. This is where database and backup checksums can make the process of determining the cause of a restore failure so valuable. Remember we talked about how restore verifies database page checksums and the backup stream checksum if they exist in the backup.

As we talked about for verifying the integrity of a backup, you can use RESTORE WITH VERIFYONLY. If this fails with the following error, you know the backup media was altered after it was written. You can't tell from this message whether the problem was with a damaged page, but you know the backup media itself was altered and the checksum is now different than when it was written.

```
Msg 3189, Level 16, State 1, Line 1
Damage to the backup set was detected.
Msg 3013, Level 16, State 1, Line 1
VERIFY DATABASE is terminating abnormally.
```

If you run the actual RESTORE DATABASE command and the damage was to a database page that has checksum values on it, you can see this type of error:

```
Msg 3183, Level 16, State 1, Line 1
RESTORE detected an error on page (1:177) in database "test_checksum" as read from
the backup set.
Msg 3013, Level 16, State 1, Line 1
RESTORE DATABASE is terminating abnormally.
```

In either case, a problem was detected with the integrity of the backup. In the case of an error with a database page checksum in the backup, the error could be due to the use of CONTINUE_AFTER_ERROR with BACKUP, or it could be that the page was damaged after the backup was written. Remember this key point about checksum. As with a standard database page checksum, the intention is for SQL Server to detect that data was modified *after* SQL Server wrote it to disk. So if you get one of the preceding two errors (assuming you didn't use BACKUP WITH CONTINUE_AFTER_ERROR) you must focus your root cause analysis of the disk system used to store the backup media.

You can always restore a different backup should you encounter these errors. But if that is not an option, you do have another choice. One of the difficult problems we had in technical support when we determined that a customer's backup was damaged was to determine the severity of the problem. In other words, if the RESTORE fails when restoring the fifth page in the backup, it stops. How can we determine whether the rest of the backup is good? In some extremely urgent cases, I even had to resort to a debugger to "skip" errors during the restore to see how badly it was damaged. SQL Server 2005 introduces a new option for RESTORE called CONTINUE_AFTER_ERROR. If you use this option, RESTORE ignores any error and proceeds to copy all data from the backup media. This way you can try to recover as much data as you possibly can. At this point, you can use DBCC CHECKDB WITH REPAIR_ALLOW_DATA_LOSS to de-allocate the damaged page(s). The damaged page from the backup was lost anyway, but now you can use the rest of the database that is still valid. One key point when using CHECKDB here: You must use the WITH TABLOCK option, too, because online repair will not provide consistent results after you have used CONTINUE_AFTER_ERROR.

Database Consistency Errors

At this point, we have covered three major areas of data recovery: issues with accessing system databases, problems accessing user databases, and backup/restore failures. Now let's talk about detecting database consistency problems during the normal database operation. I classify database consistency detection (errors) into two types:

- Runtime error, errors detected by the engine during standard usage of the database (SELECT, UPDATE, background thread such as lazywrite, and so on)
- Errors detected by DBCC CHECKDB

Handling Database Consistency Runtime Errors

The SQL Server engine contains code to perform some validation of the data (and transaction) log as you execute commands such as SELECT, INSERT, and so on against a database. The following is not a complete list of these errors, but these are the ones I anticipate you may encounter the most.

Msg 823 and *824*

The SQL Server engine has two methods to report problems reading database pages (and transaction log blocks):

- If the Windows API calls return an error when reading the page, Msg 823 is raised.
- If the Windows API call is successful, the engine performs a series of logical checks on the page (based on database options configured). If any of these checks fails, an Msg 824 is raised.

The following is an example of a Msg 823 error:

```
2005-09-07 10:51:05.16 spid18s   Error: 823, Severity: 24, State: 6.
2005-09-07 10:51:05.16 spid18s   The operating system returned
error 1117(The request could not be performed because of an I/O device error.)
to SQL Server during a read at offset 0000000000000000 in
file 'E:\EdgeTachyonDatabaseFiles\EdgePerfTachyonDB.mdf'.
```

Additional messages in the SQL Server error log and system event log may provide more detail. This is a severe system-level error condition that threatens database integrity and must be corrected immediately. Complete a full database consistency check (`DBCC CHECKDB`). This error can be caused by many factors; for more information, see SQL Server Books Online.

As you can see from the message, the SQL Server development team included the Windows error returned from the Windows API call to read the page from disk. Almost every 823 error is caused by a system problem. In this case, SQL Server reports only the error the operating system reported when reading from the file. Look at preceding the example. The text of the message even indicates there is a problem with a disk storage device. The one exception to this guideline that I've seen is error 6, "Invalid handle." In this situation, it is possible SQL Server passed an invalid file handle to the Windows API call, so this should be investigated with technical support.

If the Windows API call to read the page is successful, the following logical checks are applied in this order:

1. Is the page torn (torn page detection)?
2. Did a checksum fail when reading the page (checksum error)?
3. Is the pageid on the page different from the page we expected to read from disk (bad pageid)?

The following is an example of a `Msg 824` due to a database checksum validation check failure:

```
2006-08-25 14:38:39.34 spid53      Error: 824, Severity: 24, State: 2.
2006-08-25 14:38:39.34 spid53      SQL Server detected a logical consistency-
based I/O error: incorrect checksum (expected: 0x6cbf615; actual: 0x6dacdc1).
It occurred during a read of page (1:177) in database ID 13 at offset
0x00000000162000 in file 'C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\DATA\test_checksum.mdf'.  Additional messages in the SQL
Server error log or system event log may provide more detail. This is a
severe error condition that threatens database integrity and must be corrected
immediately. Complete a full database consistency check (DBCC CHECKDB). This
error can be caused by many factors; for more information, see SQL Server
Books Online.
```

Should you see an 824 error in the `ERRORLOG`, take immediate action, because this means the engine has detected a consistency problem in the database. The damage may be isolated to the page reported in the log, but it could go beyond that. Therefore, I recommend you run `DBCC CHECKDB` on the database as soon as possible. But which database? The server cannot provide you database context for this error, but it does provide you the filename associated with the physical filename it was reading for this page. If you can't recognize the database associated with the filename in the error, take the filename from the message and use it in the catalog view query:

```
select * from master.sys.master_files where physical_name = '<filename>'
```

At minimum, I expect `DBCC CHECKDB` to report an error for the page associated with `Msg 824`. At this point, if only a single page is damaged, you could restore from a backup (remember page-level restore here), or you can repair the database `WITH REPAIR_ALLOW_DATA_LOSS`. If more errors exist, your options may be more limited, or your data loss may be more severe. The point is, don't ignore 824 errors. It could be the tip of the iceberg of larger damage to your database.

Msg 825 (Read Retry)

While investigating a series of Exchange storage consistency problems, the Exchange development team discovered that if they were to retry failed read operations from disk, in some cases the second, third, or even fourth read attempt would succeed. So, instead of taking a beating from their customers for Exchange store corruption, they decided to add in logic to retry database read failures.

The SQL Server development team decided to adopt this same strategy for SQL Server 2005. Any read failure (823 or 824) will first be retried up to four times by the SQL Engine, before reporting the failure. If at any point the read succeeds, the following message is written to the `ERRORLOG`:

```
2005-04-27 16:06:35.98 spid346 A read of the file
'f:\mssql\mssql.1\mssql\data\stressdb2.ndf'
at offset 0x00000000302000 succeeded after failing 1 time(s) with error:
incorrect pageid (expected 4:385; actual 0:0).
Additional messages in the SQL Server error log and system event log may provide
more detail. This error condition threatens database integrity and must be
corrected. Complete a full database consistency check (DBCC CHECKDB).
This error can be caused by many factors; for more information, see SQL Server
Books Online.
```

Remember that because the retry succeeded, the user will not see any error. The only difference will be a slight delay in performance (which may or may not truly be seen by the application user). But this does not mean you do not need to address this problem. Think of this message the same as you would an 823 or 824 error, but perhaps at a slightly less-urgent pace. The engine avoided a failure, but this doesn't mean this retry will work all the time. Some disk system issue caused at least one read to fail and therefore should be investigated.

Msg 5242 and *5243* (*RecBase* Error)

The SQL Server engine also has logical checks to detect consistency problems for a row within a database page. In previous versions of SQL Server, this check would result in an assertion (a fatal error that results in a stack dump) with the expression containing the word `RecBase`. `RecBase` is the name of a class used to logically read the elements of a row on a page. See Microsoft Knowledge Base article 828337 for more information.

In SQL Server 2005, two errors were introduced to report all row consistency errors detected while reading a row as part of a standard `SELECT`, `UPDATE`, and so on, `Msg 5242` and `5243`. The following is an example of an `ERRORLOG` containing a `Msg 5242` error (with some sections omitted for brevity):

```
2006-02-18 19:33:09.40 spid51    ex_raise2: Exception raised, major=52,
minor=42, state=1, severity=22, attempting to create symptom dump
2006-02-18 19:33:09.58 spid51    Using 'dbghelp.dll' version '4.0.5'
2006-02-18 19:33:09.59 spid51    **Dump thread - spid = 51, PSS = 0x048B9278,
EC = 0x048B9280
2006-02-18 19:33:09.59 spid51    *
2006-02-18 19:33:09.59 spid51    * User initiated stack dump. This is not a server
exception dump.
2006-02-18 19:33:09.59 spid51    *
2006-02-18 19:33:09.61 spid51    ***Stack Dump being sent to C:\Program
Files\Microsoft SQL Server\MSSQL.1\MSSQL\LOG\SQLDump0001.txt
2006-02-18 19:33:09.61 spid51    *
*******************************************************************************
2006-02-18 19:33:09.61 spid51    *
2006-02-18 19:33:09.61 spid51    * BEGIN STACK DUMP:
2006-02-18 19:33:09.61 spid51    *   02/18/06 19:33:09 spid 51
2006-02-18 19:33:09.61 spid51    *
2006-02-18 19:33:09.61 spid51    * ex_raise2: Exception raised, major=52, minor=42,
state=1, severity=22
2006-02-18 19:33:09.61 spid51    *
*****************************************************************************
2006-02-18 19:33:09.61 spid51    * -------------------------------------------------
-----------------------------
2006-02-18 19:33:09.61 spid51    * Short Stack Dump
2006-02-18 19:33:09.62 spid51    77E55DEA Module(kernel32+00015DEA)
2006-02-18 19:33:09.62 spid51    02172CE4 Module(sqlservr+01172CE4)
2006-02-18 19:33:09.62 spid51    02176BA0 Module(sqlservr+01176BA0)
2006-02-18 19:33:09.62 spid51    0217674D Module(sqlservr+0117674D)
2006-02-18 19:33:09.62 spid51    01597EA2 Module(sqlservr+00597EA2)
2006-02-18 19:33:09.62 spid51    01095FFE Module(sqlservr+00095FFE)
2006-02-18 19:33:09.62 spid51    02389FB0 Module(sqlservr+01389FB0)
2006-02-18 19:33:09.62 spid51    0158E526 Module(sqlservr+0058E526)
2006-02-18 19:33:09.62 spid51    0100C833 Module(sqlservr+0000C833)
2006-02-18 19:33:09.62 spid51    0100BF0F Module(sqlservr+0000BF0F)
2006-02-18 19:33:09.62 spid51    010106EB Module(sqlservr+000106EB)
2006-02-18 19:33:09.62 spid51    01010B3A Module(sqlservr+00010B3A)
2006-02-18 19:33:09.62 spid51    01010932 Module(sqlservr+00010932)
2006-02-18 19:33:09.62 spid51    010C0E2B Module(sqlservr+000C0E2B)
2006-02-18 19:33:09.62 spid51    01210454 Module(sqlservr+00210454)
2006-02-18 19:33:09.62 spid51    013F65A8 Module(sqlservr+003F65A8)
2006-02-18 19:33:09.62 spid51    013B3E42 Module(sqlservr+003B3E42)
2006-02-18 19:33:09.62 spid51    013B35EC Module(sqlservr+003B35EC)
2006-02-18 19:33:09.62 spid51    012101C2 Module(sqlservr+002101C2)
2006-02-18 19:33:09.62 spid51    01188FC9 Module(sqlservr+00188FC9)
2006-02-18 19:33:09.62 spid51    01189021 Module(sqlservr+00189021)
2006-02-18 19:33:09.62 spid51    01330A25 Module(sqlservr+00330A25)
2006-02-18 19:33:09.62 spid51    01330421 Module(sqlservr+00330421)
2006-02-18 19:33:09.62 spid51    01332C55 Module(sqlservr+00332C55)
2006-02-18 19:33:09.62 spid51    0100889F Module(sqlservr+0000889F)
2006-02-18 19:33:09.62 spid51    010089C5 Module(sqlservr+000089C5)
2006-02-18 19:33:09.62 spid51    010086E7 Module(sqlservr+000086E7)
2006-02-18 19:33:09.62 spid51    010D764A Module(sqlservr+000D764A)
2006-02-18 19:33:09.62 spid51    010D7B71 Module(sqlservr+000D7B71)
2006-02-18 19:33:09.62 spid51    010D746E Module(sqlservr+000D746E)
```

```
2006-02-18 19:33:09.62 spid51   010D83F0 Module(sqlservr+000D83F0)
2006-02-18 19:33:09.62 spid51   781329AA Module(MSVCR80+000029AA)
2006-02-18 19:33:09.64 spid51   78132A36 Module(MSVCR80+00002A36)
2006-02-18 19:33:09.65 spid51   * ------------------------------------------
-----------------------------
2006-02-18 19:33:09.65 spid51   Stack Signature for the dump is 0xE09D5555
2006-02-18 19:33:10.48 spid51   External dump process return code 0x20000001.
External dump process returned no errors.

2006-02-18 19:33:10.58 spid51   Error: 5242, Severity: 22, State: 1.
2006-02-18 19:33:10.58 spid51   An inconsistency was detected during an
internal operation in database 'master'(ID:1) on page (1:306). Please contact
technical support. Reference number 3.
```

The error message says to contact technical support (I sure hate when our errors say that), but in reality you need to first run a `DBCC CHECKDB` in the database listed in the message. If it returns errors indicating row damage, you need to address those problems as a true database corruption problem. If `CHECKDB` does not report errors, it may be a good idea to contact technical support. Have the stack dump and `ERRORLOG` available for them to review.

Msg 605

`Msg 605` is a legacy carryover from the original SYBASE code for SQL Server. In fact, it was pretty much the only way to know about a database consistency problem aside from `DBCC CHECKDB`. This error check still remains in the SQL Server 2005 code. You should treat this error much like the row-level consistency errors 5242 and 5243. If `CHECKDB` reports errors, treat the problem as database corruption. If not, consider contacting technical support if the errors occur frequently. It could indicate some problem in the SQL Server code that should be addressed.

Access Violations

For performance reasons, the SQL Server development team cannot put in validation checks for every bit on the database page at every place in the engine code. Therefore, it is possible the engine will run into an access violation because of database corruption. How can you recognize this case? Well, because an access violation could occur at so many different places due to corruption, there is one stack dump signature I could show you to say the problem is corruption. My advice is that if you are getting access violations frequently, you should always ensure `DBCC CHECKDB` is run for all databases before pursuing the cause of the access violation as a SQL Server bug.

Failed Rollback

Another example is online rollback recovery. If the rollback of a user transaction fails, this is a critical problem. So, the engine reacts by taking the user database offline and restarting it. This allows recovery to run on the database with the hopeful effect of rolling back the transaction that couldn't roll back online. This technique actually is quite successful, because in some cases the online rollback failure is an interim problem that won't reoccur when the transactions are rolled back from the log on disk. Again, in this situation, SQL Server attempts to isolate the failure. Only the database associated with the failed rollback is affected. The rest of the engine remains intact, and other databases are not affected. As with any reliability feature, there are always exceptions. Any failure to roll back a transaction in tempdb results in a server shutdown because there is no method to run recovery on tempdb, and it is a critical shared resource for all users.

Handling *DBCC CHECKDB* Errors

I've recommended that you run DBCC CHECKDB if you encounter some of the errors described here. You may also decide to run DBCC CHECKDB at any point in time to check the consistency of your database. I presented earlier in the chapter some nice enhancements to CHECKDB to help provide more consistency checks (for example, DATA_PURITY and CHECKCATALOG). Let's talk about a strategy to address errors that you can encounter with DBCC CHECKDB. In this section, I do not go over every error that CHECKDB can report. I talk later about how you can look for information on the web about each of the errors (documented by the authors of DBCC CHECKDB, Ryan Stonecipher and Paul Randal).

Look at Summaries and Recommendation First

When you run DBCC CHECKDB and it reports errors, I recommend you do two things first:

- Look at what I call the summary messages.
- Look at the recommendation messages.

Let's look at some example output that contains errors so that I can show you what I mean:

```
DBCC results for 'checkdb_test'.
Service Broker Msg 9675, State 1: Message Types analyzed: 14.
Service Broker Msg 9676, State 1: Service Contracts analyzed: 6.
Service Broker Msg 9667, State 1: Services analyzed: 3.
Service Broker Msg 9668, State 1: Service Queues analyzed: 3.
Service Broker Msg 9669, State 1: Conversation Endpoints analyzed: 0.
Service Broker Msg 9674, State 1: Conversation Groups analyzed: 0.
Service Broker Msg 9670, State 1: Remote Service Bindings analyzed: 0.
Msg 8909, Level 16, State 1, Line 1
Table error: Object ID 0, index ID -1, partition ID 0, alloc unit ID 0 (type
Unknown), page ID (1:153) contains an incorrect page ID in its page header. The
PageId in the page header = (0:0).
Msg 8909, Level 16, State 1, Line 1
Table error: Object ID 0, index ID -1, partition ID 0, alloc unit ID 0 (type
Unknown), page ID (1:154) contains an incorrect page ID in its page header. The
PageId in the page header = (0:0).
Msg 8909, Level 16, State 1, Line 1
Table error: Object ID 0, index ID -1, partition ID 0, alloc unit ID 0 (type
Unknown), page ID (1:155) contains an incorrect page ID in its page header. The
PageId in the page header = (0:0).
Msg 8909, Level 16, State 1, Line 1
Table error: Object ID 0, index ID -1, partition ID 0, alloc unit ID 0 (type
Unknown), page ID (1:156) contains an incorrect page ID in its page header. The
PageId in the page header = (0:0).
Msg 8909, Level 16, State 1, Line 1
Table error: Object ID 0, index ID -1, partition ID 0, alloc unit ID 0 (type
Unknown), page ID (1:157) contains an incorrect page ID in its page header. The
PageId in the page header = (0:0).
Msg 8909, Level 16, State 1, Line 1
Table error: Object ID 0, index ID -1, partition ID 0, alloc unit ID 0 (type
Unknown), page ID (1:158) contains an incorrect page ID in its page header. The
PageId in the page header = (0:0).
CHECKDB found 0 allocation errors and 6 consistency errors not associated with any
single object.
DBCC results for 'sys.sysrowsetcolumns'.
There are 544 rows in 5 pages for object "sys.sysrowsetcolumns".
DBCC results for 'sys.sysrowsets'.
There are 81 rows in 1 pages for object "sys.sysrowsets".
DBCC results for 'sysallocunits'.
There are 92 rows in 1 pages for object "sysallocunits".
DBCC results for 'sys.sysfiles1'.
There are 2 rows in 1 pages for object "sys.sysfiles1".
DBCC results for 'sys.syshobtcolumns'.
There are 544 rows in 5 pages for object "sys.syshobtcolumns".
DBCC results for 'sys.syshobts'.
There are 81 rows in 1 pages for object "sys.syshobts".
DBCC results for 'sys.sysftinds'.
There are 0 rows in 0 pages for object "sys.sysftinds".
DBCC results for 'sys.sysserefs'.
There are 92 rows in 1 pages for object "sys.sysserefs".
DBCC results for 'sys.sysowners'.
There are 14 rows in 1 pages for object "sys.sysowners".
DBCC results for 'sys.sysprivs'.
There are 120 rows in 1 pages for object "sys.sysprivs".
DBCC results for 'sys.sysschobjs'.
There are 50 rows in 1 pages for object "sys.sysschobjs".
DBCC results for 'sys.syscolpars'.
There are 423 rows in 7 pages for object "sys.syscolpars".
DBCC results for 'sys.sysnsobjs'.
There are 1 rows in 1 pages for object "sys.sysnsobjs".
DBCC results for 'sys.syscerts'.
There are 0 rows in 0 pages for object "sys.syscerts".
DBCC results for 'sys.sysxprops'.
There are 0 rows in 0 pages for object "sys.sysxprops".
DBCC results for 'sys.sysscalartypes'.
There are 27 rows in 1 pages for object "sys.sysscalartypes".
DBCC results for 'sys.systypedsubobjs'.
There are 0 rows in 0 pages for object "sys.systypedsubobjs".
DBCC results for 'sys.sysidxstats'.
There are 106 rows in 1 pages for object "sys.sysidxstats".
DBCC results for 'sys.sysiscols'.
There are 218 rows in 1 pages for object "sys.sysiscols".
DBCC results for 'sys.sysbinobjs'.
There are 23 rows in 1 pages for object "sys.sysbinobjs".
DBCC results for 'sys.sysobjvalues'.
There are 104 rows in 16 pages for object "sys.sysobjvalues".
DBCC results for 'sys.sysclsobjs'.
There are 14 rows in 1 pages for object "sys.sysclsobjs".
DBCC results for 'sys.sysrowsetrefs'.
There are 0 rows in 0 pages for object "sys.sysrowsetrefs".
```

```
DBCC results for ‘sys.sysremsvcbinds’.
There are 0 rows in 0 pages for object "sys.sysremsvcbinds".
DBCC results for ‘sys.sysxmitqueue’.
There are 0 rows in 0 pages for object "sys.sysxmitqueue".
DBCC results for ‘sys.sysrts’.
There are 1 rows in 1 pages for object "sys.sysrts".
DBCC results for ‘sys.sysconvgroup’.
There are 0 rows in 0 pages for object "sys.sysconvgroup".
DBCC results for ‘sys.sysdesend’.
There are 0 rows in 0 pages for object "sys.sysdesend".
DBCC results for ‘sys.sysdercv’.
There are 0 rows in 0 pages for object "sys.sysdercv".
DBCC results for ‘sys.syssingleobjrefs’.
There are 133 rows in 1 pages for object "sys.syssingleobjrefs".
DBCC results for ‘sys.sysmultiobjrefs’.
There are 102 rows in 1 pages for object "sys.sysmultiobjrefs".
DBCC results for ‘sys.sysdbfiles’.
There are 2 rows in 1 pages for object "sys.sysdbfiles".
DBCC results for ‘sys.sysguidrefs’.
There are 0 rows in 0 pages for object "sys.sysguidrefs".
DBCC results for ‘sys.sysqnames’.
There are 91 rows in 1 pages for object "sys.sysqnames".
DBCC results for ‘sys.sysxmlcomponent’.
There are 93 rows in 1 pages for object "sys.sysxmlcomponent".
DBCC results for ‘sys.sysxmlfacet’.
There are 97 rows in 1 pages for object "sys.sysxmlfacet".
DBCC results for ‘sys.sysxmlplacement’.
There are 17 rows in 1 pages for object "sys.sysxmlplacement".
DBCC results for ‘sys.sysobjkeycrypts’.
There are 0 rows in 0 pages for object "sys.sysobjkeycrypts".
DBCC results for ‘sys.sysasymkeys’.
There are 0 rows in 0 pages for object "sys.sysasymkeys".
DBCC results for ‘sys.syssqlguides’.
There are 0 rows in 0 pages for object "sys.syssqlguides".
DBCC results for ‘sys.sysbinsubobjs’.
There are 0 rows in 0 pages for object "sys.sysbinsubobjs".
DBCC results for ‘sys.queue_messages_1977058079’.
There are 0 rows in 0 pages for object "sys.queue_messages_1977058079".
DBCC results for ‘sys.queue_messages_2009058193’.
There are 0 rows in 0 pages for object "sys.queue_messages_2009058193".
DBCC results for ‘sys.queue_messages_2041058307’.
There are 0 rows in 0 pages for object "sys.queue_messages_2041058307".
DBCC results for ‘table1’.
Msg 8928, Level 16, State 1, Line 1
Object ID 2073058421, index ID 0, partition ID 72057594038321152, alloc unit ID
72057594042318848 (type In-row data): Page (1:153) could not be processed. See
other errors for details.
Msg 8928, Level 16, State 1, Line 1
Object ID 2073058421, index ID 0, partition ID 72057594038321152, alloc unit ID
72057594042318848 (type In-row data): Page (1:154) could not be processed. See
other errors for details.
Msg 8928, Level 16, State 1, Line 1
Object ID 2073058421, index ID 0, partition ID 72057594038321152, alloc unit ID
72057594042318848 (type In-row data): Page (1:155) could not be processed. See
```

```
other errors for details.
Msg 8928, Level 16, State 1, Line 1
Object ID 2073058421, index ID 0, partition ID 72057594038321152, alloc unit ID
72057594042318848 (type In-row data): Page (1:156) could not be processed. See
other errors for details.
Msg 8928, Level 16, State 1, Line 1
Object ID 2073058421, index ID 0, partition ID 72057594038321152, alloc unit ID
72057594042318848 (type In-row data): Page (1:157) could not be processed. See
other errors for details.
Msg 8928, Level 16, State 1, Line 1
Object ID 2073058421, index ID 0, partition ID 72057594038321152, alloc unit ID
72057594042318848 (type In-row data): Page (1:158) could not be processed. See
other errors for details.
There are 88 rows in 44 pages for object "table1".
CHECKDB found 0 allocation errors and 6 consistency errors in table ‘table1’
(object ID 2073058421).
DBCC results for ‘table2’.
Msg 2511, Level 16, State 1, Line 1
Table error: Object ID 2089058478, index ID 2, partition ID 72057594038452224,
alloc unit ID 72057594042449920 (type In-row data). Keys out of order on page
(1:210), slots 157 and 158.
There are 1000 rows in 3 pages for object “table2".
CHECKDB found 0 allocation errors and 1 consistency errors in table ‘table2’
(object ID 2089058478).
CHECKDB found 0 allocation errors and 13 consistency errors in database
‘checkdb_test’.
repair_allow_data_loss is the minimum repair level for the errors found by
DBCC CHECKDB (checkdb_test).
DBCC execution completed. If DBCC printed error messages, contact your system
administrator.
```

Go to the bottom of the output and look at the summary and recommendation messages for the database:

```
CHECKDB found 0 allocation errors and 13 consistency errors in database
‘checkdb_test’.
repair_allow_data_loss is the minimum repair level for the errors found by
DBCC CHECKDB (checkdb_test).
```

You know from this message that 13 errors related to the consistency of pages or indexes have been detected (as opposed to allocation errors that are specific to problems with allocation structures found by the `CHECKALLOC` phase). You also know that in order to repair *all* 13 errors, you must use the `REPAIR_ALLOW_DATA_LOSS` repair option. You could just move forward with this recommendation and check the results. However, if you want to know what the repair recommendation is for each table found to have problems, you could check each one. Which tables do I check? Look at the summary messages for each table:

```
CHECKDB found 0 allocation errors and 6 consistency errors not associated with any
single object.
CHECKDB found 0 allocation errors and 6 consistency errors in table ‘table1’
(object ID 2073058421).
```

```
CHECKDB found 0 allocation errors and 1 consistency errors in table 'table2'
(object ID 2089058478).
```

First, you should know that CHECKDB won't produce this summary message for a table unless it detects problems with that table. Second, notice the first summary message says "not associated with any single object." This message indicates that CHECKDB found errors associated with pages that don't appear to belong to a known table.

In this situation, what is happening is that CHECKDB is in a way "double-reporting" errors, but that is to your advantage in this situation. Let's look at the errors "not associated with any single object."

```
Msg 8909, Level 16, State 1, Line 1
Table error: Object ID 0, index ID -1, partition ID 0, alloc unit ID 0 (type
Unknown), page ID (1:153) contains an incorrect page ID in its page header.
The PageId in the page header = (0:0).
```

This message indicates a problem with the page ID as found on the page header. The page ID was expected to be 153, but on the page it was found to be 0:0. The routine that detects this problem is used throughout the CHECKDB code to read a page. In this routine, the table associated with the page is found by looking up the allocation unit ID on the page header. As you can see from the error message, that value is 0, and therefore CHECKDB doesn't know what table this page belongs to.

Now look at the errors associated with table1:

```
Msg 8928, Level 16, State 1, Line 1
Object ID 2073058421, index ID 0, partition ID 72057594038321152, alloc unit
ID 72057594042318848 (type In-row data): Page (1:153) could not be processed.
See other errors for details.
```

If you look closely, you will notice this error is for the same page as the one I just talked about for 8909. In fact, all the pages in errors 8909 and 8928 are the same. Why does the server report two different errors for the same page? I just mentioned that the 8909 error is reported in a general routine to read a page. This routine is self-contained and so relies on finding the object from the allocation unit id on the page. The code that raises the 8928 error is part of a routine that is checking "facts" about pages associated with an index. (In this case, it was actually just a heap for table1.) This code already "knows" what index it was checking (table1) and knows what page it was trying to read, so it can accurately report that it had a problem reading a page and what table it thought the page was associated with. You will notice this error says, "See other errors for details." What this message is saying is this: "I've detected a problem with this page, but there should be another error describing the actual problem." In this case, the other error is 8909, but it is not associated with table1 for the reasons I just described. Perhaps by now you can guess the problem in this scenario with some of these pages. A quick glance at page 1:153 using DBCC PAGE would tell you. (I didn't include the entire output because it is not important for this situation.)

```
dbcc traceon(3604)
go
dbcc page(10, 1, 153, 2)
```

```
go
DBCC execution completed. If DBCC printed error messages, contact your system
administrator.
PAGE: (0:0)

BUFFER:

BUF @0x02BF096C
bpage = 0x05052000          bhash = 0x00000000          bpageno = (1:153)
bdbid = 10              breferences = 1            bUse1 = 50447
bstat = 0xc00809           blog = 0x32159             bnext = 0x00000000
PAGE HEADER:

Page @0x05052000
m_pageId = (0:0)            m_headerVersion = 0          m_type = 0
m_typeFlagBits = 0x0          m_level = 0             m_flagBits = 0x0
m_objId (AllocUnitId.idObj) = 0   m_indexId (AllocUnitId.idInd) = 0  Metadata:
AllocUnitId = 0
Metadata: PartitionId = 0      Metadata: IndexId = -1       Metadata: ObjectId = 0
m_prevPage = (0:0)           m_nextPage = (0:0)           pminlen = 0
m_slotCnt = 0             m_freeCnt = 0             m_freeData = 0
m_reservedCnt = 0           m_lsn = (0:0:0)            m_xactReserved = 0
m_xdesId = (0:0)            m_ghostRecCnt = 0           m_tornBits = 0
Allocation Status
GAM (1:2) = ALLOCATED         SGAM (1:3) = NOT ALLOCATED
PFS (1:1) = 0x64 MIXED_EXT ALLOCATED 100_PCT_FULL              DIFF (1:6) = CHANGED
ML (1:7) = NOT MIN_LOGGED
DATA:

Memory Dump @0x44C7C000
44C7C000:  00000000 00000000 00000000 00000000 †................
44C7C010:  00000000 00000000 00000000 00000000 †................
44C7C020:  00000000 00000000 00000000 00000000 †................
44C7C030:  00000000 00000000 00000000 00000000 †................
44C7C040:  00000000 00000000 00000000 00000000 †................
44C7C050:  00000000 00000000 00000000 00000000 †................
44C7C060:  1000a80f 02000000 4d595441 42202020 †........MYTAB
44C7C070:  20202020 20202020 20202020 20202020 †
44C7C080:  20202020 20202020 20202020 20202020 †
44C7C090:  20202020 20202020 20202020 20202020 †
44C7C0A0:  20202020 20202020 20202020 20202020 †
44C7C0B0:  20202020 20202020 20202020 20202020 †
44C7C0C0:  20202020 20202020 20202020 20202020 †
44C7C0D0:  20202020 20202020 20202020 20202020 †


44C7C0E0:  20202020 20202020 20202020 20202020 †
```

The first 96 bytes of any page is the page header. I think you can easily see that just about the entire header has all values of 0. If in this situation the page ID were the only problem with the page, `Msg 8909` would have correctly shown up as a problem for table1. So, in this example, you can see that of the 13 errors, six of them are reported more than once. In reality, therefore, only seven pages have problems from this database.

Let's take a look at the errors for table2:

```
Msg 2511, Level 16, State 1, Line 1
Table error: Object ID 2089058478, index ID 2, partition ID 72057594038452224,
alloc unit ID 72057594042449920 (type In-row data). Keys out of order on page
(1:210), slots 157 and 158.
```

In this case, there is only one error, and as you can see it is associated with a nonclustered index (index ID 2). You saw that the 8928 errors were associated with data pages (index ID 0). Can you guess what the recommended repair level for table2 would be? Because the output may not always be this simple, let's use `DBCC CHECKTABLE` on each table to know for sure.

Here are the results for `CHECKTABLE` for table1:

```
Msg 8909, Level 16, State 1, Line 1
Table error: Object ID 0, index ID -1, partition ID 0, alloc unit ID 0 (type
Unknown), page ID (1:153) contains an incorrect page ID in its page header. The
PageId in the page header = (0:0).
Msg 8909, Level 16, State 1, Line 1
Table error: Object ID 0, index ID -1, partition ID 0, alloc unit ID 0 (type
Unknown), page ID (1:154) contains an incorrect page ID in its page header. The
PageId in the page header = (0:0).
Msg 8909, Level 16, State 1, Line 1
Table error: Object ID 0, index ID -1, partition ID 0, alloc unit ID 0 (type
Unknown), page ID (1:155) contains an incorrect page ID in its page header. The
PageId in the page header = (0:0).
Msg 8909, Level 16, State 1, Line 1
Table error: Object ID 0, index ID -1, partition ID 0, alloc unit ID 0 (type
Unknown), page ID (1:156) contains an incorrect page ID in its page header. The
PageId in the page header = (0:0).
Msg 8909, Level 16, State 1, Line 1
Table error: Object ID 0, index ID -1, partition ID 0, alloc unit ID 0 (type
Unknown), page ID (1:157) contains an incorrect page ID in its page header. The
PageId in the page header = (0:0).
Msg 8909, Level 16, State 1, Line 1
Table error: Object ID 0, index ID -1, partition ID 0, alloc unit ID 0 (type
Unknown), page ID (1:158) contains an incorrect page ID in its page header. The
PageId in the page header = (0:0).
CHECKTABLE found 0 allocation errors and 6 consistency errors not associated with
any single object.
DBCC results for ‘table1’.
Msg 8928, Level 16, State 1, Line 1
Object ID 2073058421, index ID 0, partition ID 72057594038321152, alloc unit ID
72057594042318848 (type In-row data): Page (1:153) could not be processed. See
other errors for details.
Msg 8928, Level 16, State 1, Line 1
Object ID 2073058421, index ID 0, partition ID 72057594038321152, alloc unit ID
72057594042318848 (type In-row data): Page (1:154) could not be processed. See
other errors for details.
Msg 8928, Level 16, State 1, Line 1
Object ID 2073058421, index ID 0, partition ID 72057594038321152, alloc unit ID
72057594042318848 (type In-row data): Page (1:155) could not be processed. See
other errors for details.
```

```
Msg 8928, Level 16, State 1, Line 1
Object ID 2073058421, index ID 0, partition ID 72057594038321152, alloc unit ID
72057594042318848 (type In-row data): Page (1:156) could not be processed. See
other errors for details.
Msg 8928, Level 16, State 1, Line 1
Object ID 2073058421, index ID 0, partition ID 72057594038321152, alloc unit ID
72057594042318848 (type In-row data): Page (1:157) could not be processed. See
other errors for details.
Msg 8928, Level 16, State 1, Line 1
Object ID 2073058421, index ID 0, partition ID 72057594038321152, alloc unit ID
72057594042318848 (type In-row data): Page (1:158) could not be processed. See
other errors for details.
There are 88 rows in 44 pages for object "table1".
CHECKTABLE found 0 allocation errors and 6 consistency errors in table 'table1'
(object ID 2073058421).
repair_allow_data_loss is the minimum repair level for the errors found by
DBCC CHECKTABLE (checkdb_test.dbo.table1).

DBCC execution completed. If DBCC printed error messages, contact your system
administrator.
```

Notice the repair recommendation is still `REPAIR_ALLOW_DATA_LOSS`. Because these six pages are all data pages, a repair will actually result in lost data, because the pages will be de-allocated.

Here are the results of `CHECKTABLE` for table2:

```
DBCC results for 'table2'.
Msg 2511, Level 16, State 1, Line 1
Table error: Object ID 2089058478, index ID 2, partition ID 72057594038452224,
alloc unit ID 72057594042449920 (type In-row data). Keys out of order on page
(1:210), slots 157 and 158.
There are 1000 rows in 3 pages for object "table2".
CHECKTABLE found 0 allocation errors and 1 consistency errors in table
'table2' (object ID 2089058478).
repair_rebuild is the minimum repair level for the errors found by DBCC
CHECKTABLE (checkdb_test.dbo.table2).
```

`DBCC` execution completed. If `DBCC` printed error messages, contact your system administrator.

You can see that the repair recommendation is `REPAIR_REBUILD`, which means a rebuild of an index will correct any errors, and no data loss should occur.

Let's see what the output of `CHECKDB` looks like in this situation (because `CHECKDB` will report messages on what actions it took to repair any error it can fix):

```
DBCC results for 'checkdb_test'.
Service Broker Msg 9675, State 1: Message Types analyzed: 14.
Service Broker Msg 9676, State 1: Service Contracts analyzed: 6.
Service Broker Msg 9667, State 1: Services analyzed: 3.
Service Broker Msg 9668, State 1: Service Queues analyzed: 3.
Service Broker Msg 9669, State 1: Conversation Endpoints analyzed: 0.
Service Broker Msg 9674, State 1: Conversation Groups analyzed: 0.
Service Broker Msg 9670, State 1: Remote Service Bindings analyzed: 0.
Msg 8909, Level 16, State 1, Line 1
Table error: Object ID 0, index ID -1, partition ID 0, alloc unit ID 0 (type
Unknown), page ID (1:153) contains an incorrect page ID in its page header. The
PageId in the page header = (0:0).
    The error has been repaired.
Msg 8909, Level 16, State 1, Line 1
Table error: Object ID 0, index ID -1, partition ID 0, alloc unit ID 0 (type
Unknown), page ID (1:154) contains an incorrect page ID in its page header. The
PageId in the page header = (0:0).
    The error has been repaired.
Msg 8909, Level 16, State 1, Line 1
Table error: Object ID 0, index ID -1, partition ID 0, alloc unit ID 0 (type
Unknown), page ID (1:155) contains an incorrect page ID in its page header. The
PageId in the page header = (0:0).
    The error has been repaired.
Msg 8909, Level 16, State 1, Line 1
Table error: Object ID 0, index ID -1, partition ID 0, alloc unit ID 0 (type
Unknown), page ID (1:156) contains an incorrect page ID in its page header. The
PageId in the page header = (0:0).
    The error has been repaired.
Msg 8909, Level 16, State 1, Line 1
Table error: Object ID 0, index ID -1, partition ID 0, alloc unit ID 0 (type
Unknown), page ID (1:157) contains an incorrect page ID in its page header. The
PageId in the page header = (0:0).
    The error has been repaired.
Msg 8909, Level 16, State 1, Line 1
Table error: Object ID 0, index ID -1, partition ID 0, alloc unit ID 0 (type
Unknown), page ID (1:158) contains an incorrect page ID in its page header. The
PageId in the page header = (0:0).
    The error has been repaired.
CHECKDB found 0 allocation errors and 6 consistency errors not associated with any
single object.
CHECKDB fixed 0 allocation errors and 6 consistency errors not associated with any
single object.
DBCC results for ‘sys.sysrowsetcolumns’.
There are 544 rows in 5 pages for object "sys.sysrowsetcolumns".
DBCC results for ‘sys.sysrowsets’.
There are 81 rows in 1 pages for object "sys.sysrowsets".
DBCC results for ‘sysallocunits’.
There are 92 rows in 1 pages for object "sysallocunits".
DBCC results for ‘sys.sysfiles1’.
There are 2 rows in 1 pages for object "sys.sysfiles1".
DBCC results for ‘sys.syshobtcolumns’.
There are 544 rows in 5 pages for object "sys.syshobtcolumns".
DBCC results for ‘sys.syshobts’.
There are 81 rows in 1 pages for object "sys.syshobts".
DBCC results for ‘sys.sysftinds’.
There are 0 rows in 0 pages for object "sys.sysftinds".
DBCC results for ‘sys.sysserefs’.
There are 92 rows in 1 pages for object "sys.sysserefs".
DBCC results for ‘sys.sysowners’.
There are 14 rows in 1 pages for object "sys.sysowners".
DBCC results for ‘sys.sysprivs’.
```

```
There are 120 rows in 1 pages for object "sys.sysprivs".
DBCC results for 'sys.sysschobjs'.
There are 50 rows in 1 pages for object "sys.sysschobjs".
DBCC results for 'sys.syscolpars'.
There are 423 rows in 7 pages for object "sys.syscolpars".
DBCC results for 'sys.sysnsobjs'.
There are 1 rows in 1 pages for object "sys.sysnsobjs".
DBCC results for 'sys.syscerts'.
There are 0 rows in 0 pages for object "sys.syscerts".
DBCC results for 'sys.sysxprops'.
There are 0 rows in 0 pages for object "sys.sysxprops".
DBCC results for 'sys.sysscalartypes'.
There are 27 rows in 1 pages for object "sys.sysscalartypes".
DBCC results for 'sys.systypedsubobjs'.
There are 0 rows in 0 pages for object "sys.systypedsubobjs".
DBCC results for 'sys.sysidxstats'.
There are 116 rows in 2 pages for object "sys.sysidxstats".
DBCC results for 'sys.sysiscols'.
There are 228 rows in 1 pages for object "sys.sysiscols".
DBCC results for 'sys.sysbinobjs'.
There are 23 rows in 1 pages for object "sys.sysbinobjs".
DBCC results for 'sys.sysobjvalues'.
There are 114 rows in 17 pages for object "sys.sysobjvalues".
DBCC results for 'sys.sysclsobjs'.
There are 14 rows in 1 pages for object "sys.sysclsobjs".
DBCC results for 'sys.sysrowsetrefs'.
There are 0 rows in 0 pages for object "sys.sysrowsetrefs".
DBCC results for 'sys.sysremsvcbinds'.
There are 0 rows in 0 pages for object "sys.sysremsvcbinds".
DBCC results for 'sys.sysxmitqueue'.
There are 0 rows in 0 pages for object "sys.sysxmitqueue".
DBCC results for 'sys.sysrts'.
There are 1 rows in 1 pages for object "sys.sysrts".
DBCC results for 'sys.sysconvgroup'.
There are 0 rows in 0 pages for object "sys.sysconvgroup".
DBCC results for 'sys.sysdesend'.
There are 0 rows in 0 pages for object "sys.sysdesend".
DBCC results for 'sys.sysdercv'.
There are 0 rows in 0 pages for object "sys.sysdercv".
DBCC results for 'sys.syssingleobjrefs'.
There are 133 rows in 1 pages for object "sys.syssingleobjrefs".
DBCC results for 'sys.sysmultiobjrefs'.
There are 102 rows in 1 pages for object "sys.sysmultiobjrefs".
DBCC results for 'sys.sysdbfiles'.
There are 2 rows in 1 pages for object "sys.sysdbfiles".
DBCC results for 'sys.sysguidrefs'.
There are 0 rows in 0 pages for object "sys.sysguidrefs".
DBCC results for 'sys.sysqnames'.
There are 91 rows in 1 pages for object "sys.sysqnames".
DBCC results for 'sys.sysxmlcomponent'.
There are 93 rows in 1 pages for object "sys.sysxmlcomponent".
DBCC results for 'sys.sysxmlfacet'.
There are 97 rows in 1 pages for object "sys.sysxmlfacet".
DBCC results for 'sys.sysxmlplacement'.
```

```
There are 17 rows in 1 pages for object "sys.sysxmlplacement".
DBCC results for 'sys.sysobjkeycrypts'.
There are 0 rows in 0 pages for object "sys.sysobjkeycrypts".
DBCC results for 'sys.sysasymkeys'.
There are 0 rows in 0 pages for object "sys.sysasymkeys".
DBCC results for 'sys.syssqlguides'.
There are 0 rows in 0 pages for object "sys.syssqlguides".
DBCC results for 'sys.sysbinsubobjs'.
There are 0 rows in 0 pages for object "sys.sysbinsubobjs".
DBCC results for 'sys.queue_messages_1977058079'.
There are 0 rows in 0 pages for object "sys.queue_messages_1977058079".
DBCC results for 'sys.queue_messages_2009058193'.
There are 0 rows in 0 pages for object "sys.queue_messages_2009058193".
DBCC results for 'sys.queue_messages_2041058307'.
There are 0 rows in 0 pages for object "sys.queue_messages_2041058307".
DBCC results for 'table1'.
Repair: The page (1:153) has been deallocated from object ID 2073058421, index ID
0, partition ID 72057594038321152, alloc unit ID 72057594042318848 (type In-row
data).
Repair: The page (1:154) has been deallocated from object ID 2073058421, index ID
0, partition ID 72057594038321152, alloc unit ID 72057594042318848 (type In-row
data).
Repair: The page (1:155) has been deallocated from object ID 2073058421, index ID
0, partition ID 72057594038321152, alloc unit ID 72057594042318848 (type In-row
data).
Repair: The page (1:156) has been deallocated from object ID 2073058421, index ID
0, partition ID 72057594038321152, alloc unit ID 72057594042318848 (type In-row
data).
Repair: The page (1:157) has been deallocated from object ID 2073058421, index ID
0, partition ID 72057594038321152, alloc unit ID 72057594042318848 (type In-row
data).
Repair: The page (1:158) has been deallocated from object ID 2073058421, index ID
0, partition ID 72057594038321152, alloc unit ID 72057594042318848 (type In-row
data).
Msg 8928, Level 16, State 1, Line 1
Object ID 2073058421, index ID 0, partition ID 72057594038321152, alloc unit ID
72057594042318848 (type In-row data): Page (1:153) could not be processed. See
other errors for details.
    The error has been repaired.
Msg 8928, Level 16, State 1, Line 1
Object ID 2073058421, index ID 0, partition ID 72057594038321152, alloc unit ID
72057594042318848 (type In-row data): Page (1:154) could not be processed. See
other errors for details.
    The error has been repaired.
Msg 8928, Level 16, State 1, Line 1
Object ID 2073058421, index ID 0, partition ID 72057594038321152, alloc unit ID
72057594042318848 (type In-row data): Page (1:155) could not be processed. See
other errors for details.
    The error has been repaired.
Msg 8928, Level 16, State 1, Line 1
Object ID 2073058421, index ID 0, partition ID 72057594038321152, alloc unit ID
72057594042318848 (type In-row data): Page (1:156) could not be processed. See
other errors for details.
    The error has been repaired.
```

```
Msg 8928, Level 16, State 1, Line 1
Object ID 2073058421, index ID 0, partition ID 72057594038321152, alloc unit ID
72057594042318848 (type In-row data): Page (1:157) could not be processed. See
other errors for details.
     The error has been repaired.
Msg 8928, Level 16, State 1, Line 1
Object ID 2073058421, index ID 0, partition ID 72057594038321152, alloc unit ID
72057594042318848 (type In-row data): Page (1:158) could not be processed. See
other errors for details.
     The error has been repaired.
There are 88 rows in 44 pages for object "table1".
CHECKDB found 0 allocation errors and 6 consistency errors in table ‘table1’
(object ID 2073058421).
CHECKDB fixed 0 allocation errors and 6 consistency errors in table ‘table1’
(object ID 2073058421).
DBCC results for ‘table2’.
Repair: The Nonclustered index successfully rebuilt for the object "dbo.table2,
PK_ _table2_ _7D78A4E7" in database "checkdb_test".
Msg 8945, Level 16, State 1, Line 1
Table error: Object ID 2089058478, index ID 2 will be rebuilt.
     The error has been repaired.
Msg 2511, Level 16, State 1, Line 1
Table error: Object ID 2089058478, index ID 2, partition ID 72057594038452224,
alloc unit ID 72057594042449920 (type In-row data). Keys out of order on page
(1:210), slots 157 and 158.
     The error has been repaired.
There are 1000 rows in 3 pages for object "table2".
CHECKDB found 0 allocation errors and 1 consistency errors in table ‘table2’
(object ID 2089058478).
CHECKDB fixed 0 allocation errors and 1 consistency errors in table ‘table2’
(object ID 2089058478).
CHECKDB found 0 allocation errors and 13 consistency errors in database
‘checkdb_test’.
CHECKDB fixed 0 allocation errors and 13 consistency errors in database
‘checkdb_test’.
DBCC execution completed. If DBCC printed error messages, contact your system
administrator.
```

In this case, I’ve highlighted the important messages that are added as part of repair:

```
Repair: The page (1:153) has been deallocated from object ID 2073058421, index ID
0, partition ID 72057594038321152, alloc unit ID 72057594042318848 (type In-row
data).
Repair: The page (1:154) has been deallocated from object ID 2073058421, index ID
0, partition ID 72057594038321152, alloc unit ID 72057594042318848 (type In-row
data).
Repair: The page (1:155) has been deallocated from object ID 2073058421, index ID
0, partition ID 72057594038321152, alloc unit ID 72057594042318848 (type In-row
data).
Repair: The page (1:156) has been deallocated from object ID 2073058421, index ID
0, partition ID 72057594038321152, alloc unit ID 72057594042318848 (type In-row
data).
```

```
Repair: The page (1:157) has been deallocated from object ID 2073058421, index
ID 0, partition ID 72057594038321152, alloc unit ID 72057594042318848 (type
In-row data).
Repair: The page (1:158) has been deallocated from object ID 2073058421, index
ID 0, partition ID 72057594038321152, alloc unit ID 72057594042318848 (type
In-row data).
```

You can see from these messages that the method to repair these pages is to de-allocate them. There is no method to retrieve any data that could have been on these pages:

```
Repair: The Nonclustered index successfully rebuilt for the object "dbo.table2,
PK_ _table2_ _7D78A4E7" in database "checkdb_test".
Msg 8945, Level 16, State 1, Line 1
Table error: Object ID 2089058478, index ID 2 will be rebuilt.
```

This is the error for table2. You can see that the only method needed to repair the table is to rebuild the nonclustered index:

```
CHECKDB found 0 allocation errors and 13 consistency errors in database
‘checkdb_test’.
CHECKDB fixed 0 allocation errors and 13 consistency errors in database
‘checkdb_test’.
```

This is the most important message to look for. Was the repair successful? If the number of messages *fixed* is less than those *found,* the repair couldn’t fix all the problems.

Repair Versus Restore

How do you know whether you should restore from a database backup or use repair? One consideration is what I’ve described in the section “Best Practices”: the primary feature for a recovery of data for the SQL Server product is `RESTORE`. Repair has been a great feature of `DBCC CHECKDB`, but you should use it only for emergency purposes. We talk shortly about what `REPAIR_ALLOW_DATA_LOSS` really means, but you should know that in most cases it does mean you will probably lose data. Here is the problem. Aside from knowing what table is affected by the data loss, you won’t know what rows you lost. It is now your responsibility to understand what level of logical consistency you now have in your database.

If you have the proper backup strategy in place, you should almost never have to rely on repair. But the functionality is still there in case you need it. (Perhaps you couldn’t predict that a disk that holds backups and is always reliable has an unexpected failure. Of course, I think you should have a plan for even this scenario.) You will find that if you call Microsoft PSS to talk about a situation where you have a damaged database and want to talk about whether repair will fix the problem, instead of just helping you run `REPAIR`, PSS will almost always talk to you about using a backup first.

What Does Each Error Mean?

Each error produced by `DBCC CHECKDB` is documented on the Events and Errors Message Center. Some of these errors that apply to both SQL Server 2000 and 2005 are documented under SQL Server 2000. Here is an example of the documentation of an error that can be encountered in `CHECKDB`:

Details

Product:	SQL Server
Event ID:	2531
Source:	MSSQLServer
Version:	9.00.1281.60
Symbolic Name:	DBCC_BTREE_SIBLING_LEVEL_MISMATCH
Message:	Table error: object ID %d, index ID %d, partition ID %I64d, alloc unit ID %I64d (type %.*ls) B-tree level mismatch, page %S_PGID. Level %d does not match level %d from the previous %S_PGID.

Explanation

Two pages are linked as immediate neighbors on a level of a B-tree. The level, LEVEL2, on the right page, P_ID2, does not match the level, LEVEL1, on the left page, P_ID1.

To determine which page is incorrect, examine the surrounding pages and the contents of the two pages in question. Also, look for `MSSQLEngine_8931` errors, which indicate B-tree parent-child-level mismatches.

Possible Causes

This error can be caused by one of the following problems:

- A random page corruption.
- A bug in the B-tree manager.
- If LEVEL1 and LEVEL2 are 0 or 1 and the index is a clustered index, there might be a bug in the Access Methods code that determines page levels. In ssVersion2005, for a clustered index, page levels progress from 0, 1, 2 to X, but in ssVersion2000, for a clustered index, page levels progress from 0, 0, 1, 2 to X, where X is the maximum depth of the B-tree.

User Action

Run hardware diagnostics and correct any problems. Also examine the Microsoft Windows system and application logs and the SQL Server error log to see whether the error occurred as the result of hardware failure. Fix any hardware-related problems that are contained in the logs.

If you have persistent data corruption problems, try to swap out different hardware components to isolate the problem. Check to make sure that the system does not have write caching enabled on the disk controller. If you suspect write caching to be the problem, contact your hardware vendor.

Finally, you might find it useful to switch to a new hardware system. This switch may include reformatting the disk drives and reinstalling the operating system.

Results of Running Repair Options

Repair will rebuild the index.

This is the general format for each message:

- An explanation of what the error means with regard to what type of data is damaged
- Possible causes of the problem
- What actions you can take
- What repair will do to correct the problem

The actions to take seem general (check the hardware and so forth), but you will see when I talk about finding the root cause of corruption that these actions make sense for many errors detected by `CHECKDB`.

What Does *REPAIR_ALLOW_DATA_LOSS* Really Mean?

This repair option means that `CHECKDB` has detected at least one error that could result in data loss. I've already talked about how this may not apply to every error detected. This is why the repair recommendation message says the *minimum* repair level:

```
repair_allow_data_loss is the minimum repair level…
```

The method for repair to correct a problem recommended by `REPAIR_ALLOW_DATA_LOSS` usually is to de-allocate the page or groups of pages (extent) based on the error. In some scenarios, a data row could be deleted rather than the entire page, but they are limited (for example, if the LOB pages associated with the row are damaged and must be de-allocated). If any part of the structure of the data row itself is damaged, the entire page must be de-allocated.

I know of two situations in which the recommendation is `REPAIR_ALLOW_DATA_LOSS` but repair actually can correct the problem without a loss of data.

Damaged index page

If `DBCC CHECKDB` encounters an error such as a checksum failure for a nonclustered or nonleaf page of a clustered index, it recommends `REPAIR_ALLOW_DATA_LOSS`. This is because it cannot trust the allocation unit ID information on the page to know for sure it is a nonclustered index page. So it must de-allocate the page. However, the repair logic in this case also rebuilds the index so that the result is simply to rebuild the nonclustered index.

PFS free space error

You may encounter the following error when running `DBCC CHECKDB`:

```
Msg 8914, Level 16, State 1, Line 1
Incorrect PFS free space information for page (1:128) in object ID 60, index
ID 1, partition ID 281474980642816, alloc unit ID 71776119065149440 (type LOB
data). Expected value  0_PCT_FULL, actual value 100_PCT_FULL.
```

In this case, the `CHECKDB` code recommends `REPAIR_ALLOW_DATA_LOSS` but repair simply fixes the free space information in the PFS page with no data loss occurring as a result.

Root Cause Analysis Before Recovering

It is perfectly understandable for you in situations where `CHECKDB` reports an error to want to correct the problem as soon as possible, either by restoring from a backup or by using repair. However, if you want to have the ability to determine the possible root cause of the problem, I recommend you take the following steps:

1. Try to back up the damaged database, and keep it before restoring or repair.

   You may not have time to do this. That is understandable in production situations. But if you have the time and disk space, please take this step. If you contact Microsoft technical support, this is one of the things they will want you to do.

2. Take the time to investigate any possible hardware or system problems.
3. Use some of the techniques I describe later in this chapter for root cause analysis to check your system.

What if Repair Doesn't Work?

There are some errors that `CHECKDB` cannot fix:

- Critical system table damage
- PFS page damage
- Data purity errors

So what if you run repair and it doesn't fix all errors? In previous versions, I would tell you to just try it again. There were some situations where repair had to be run more than once to clean up all errors. Those have been corrected for SQL Server 2005. If repair simply can't fix all of your errors, your only choice is to copy all data possible from the database using `BCP` or `SELECT` and move it to another database.

Copying Data Versus Repair

Are there situations where you should consider trying to copy data using `BCP` or `SELECT` *before* you try to use repair? The only example I know of is where you may want to try and copy valid rows from a page where only a single row or a few rows are damaged. Because a damaged row would cause repair to de-allocate the page, you may want to see whether you can copy valid rows from the page except for these damaged rows. Of course, this makes sense only if these specific rows are that crucial to your business.

The only method to do this is to force the use of a nonclustered index to select specific rows (using the index hint in the `SELECT` statement) based on values using the index to obtain rows "around" the damaged row(s). This assumes the nonclustered index itself is intact. If you don't have a valid nonclustered index, you cannot create one in this situation. This technique can work because when the engine finds a specific row based on using the nonclustered index (assuming you have the right `WHERE` criteria), it can avoid scanning unnecessary rows. I've included an exercise at the end of this chapter for you to walk through an example of how to do this.

Find the Root Cause of Corruption: The Checklist

So, let's say you encounter errors from `DBCC CHECKDB`, but this is the first time you have ever seen it. You decide you don't want to invest much time in finding the cause. A restore of the database resolves the problem, it didn't take long, and you want to move on to other problems you need to solve. That may be fine, but what if at some point a few weeks you run `CHECKDB` and the same database has errors again? What if a different database is now damaged?

I've put together here a checklist of what I as a PSS engineer over the years use when a customer wants me to investigate the root cause of a database corruption problem as evidence of reoccurring errors from `CHECKDB` (or perhaps checksum errors on various pages).

As you read through this list, keep in mind one key fact about root cause of database corruption: *Almost all database corruption situations are caused by a problem with the underlying disk system* (drivers, controllers, firmware, disk, and so on). I'm not just telling you this because I work for Microsoft. I give you this observation based on my experience of seeing many customer reports of database corruption through the years of supporting SQL Server 7.0, 2000, and now 2005. As I go through this list, I indicate whether a particular piece of information you collect could point to a system problem or perhaps some other possible cause.

You should also know that in my experience many corruption problems go unsolved. Typically this is because

- There may not be an obvious sign of why the corruption occurred (for example, no system event log entry for a hardware problem).
- Many corruption problems do not reoccur.
- The diagnostics and time required to find the cause can be expensive.

Time will tell, but I'm hopeful that the database checksum feature for SQL Server 2005 will help improve the first reason. This is because a checksum error (`Msg 824`) would indicate some alteration to a database page had occurred since the engine wrote the page to disk.

As you read through this checklist, keep in mind that just about any information I discuss for data collection can be obtained using the `SQLDIAG.EXE` utility. This is the primary data collection tool to ensure all information is captured at a consistent point in time.

Keep Track of Problem Details and History

The root cause of any problem is always difficult if you don't have the right information and, in some cases, the history behind the problem. For reoccurring corruption problems, this is important.

Review ERRORLOG and Event Log

I hope you know how important it is to have the SQL `ERRORLOG` and System/Application event logs when working on problems of this magnitude. The typical information I look for when reviewing an `ERRORLOG` for this type of problem is as follows:

- "Last known good" for `CHECKDB` of databases
- Summary message on errors and repair
- Any stack dumps or critical errors (such as `Msg 824`)
- The use of any extended procedures or sp_OA COM objects (memory scribblers)

The event log is important, too. First, perhaps your `ERRORLOG` files have wrapped, but important past information in the event log has not been cleared.

Next, the System event log may contain information about possible disk, hardware, or Windows system problems. Table 2-2 contains a few common event log entries seen by PSS in the context of corruption cases, with pointers on content to read about each type of error.

TABLE 2-2 Event log entries seen by PSS

Source	Error	Notes
<any>	The device, \Device\Scsi\cpqcissm1, did not respond within the timeout period.	See KB 259237 and 154690.
Disk	The driver detected a controller error on \Device\Harddisk4\DR4.	See KBs 259237 and 154690.
Disk	The device, \Device\Harddisk14\DR14, is not ready for access yet.	See KB 259237.
SaveDump	The computer has rebooted from Bluescreen...a bugcheck. The bugcheck was: ...	Engage Windows Support.
Disk	An error was detected on device \Device\Harddisk3\DR3 during a paging operation.	This error indicates an I/O error during a hard page fault. Discussed in KBs 304415 and 305547.
ClusSvc	Cluster disk resource Disk J:: is corrupt. Running ChkDsk /F to repair problems.	KB 259237: "can be the result of SCSI host adapter configuration issues" or a malfunctioning device. Also see 311081 and 259237.
Ntfs	The file system structure on the disk is corrupt and unusable. Please run the chkdsk utility on the volume F:.	Note that there is at least one case (320866) where this error is erroneously raised.
Disk	Data was recovered using error correction code on device \Device\Harddisk5\DR5.	
EventLog	The previous system shutdown at 9:45:36 AM on 9/5/2004 was unexpected.	Typically indicates a hard server cycle after a hang or a blue screen. Could also indicate something more mundane such as a power failure if the system isn't protected by UPS.
Ftdisk	{Lost Delayed-Write Data} The system was attempting to transfer file data from buffers to \Device\HarddiskVolume4. The write operation failed, and only some of the data may have been written to the file.	Indicates a failed I/O request. Discussed in KBs 311081 and 304415. Could be anything from a firmware bug to faulty SCSI cables.

Perform Hardware Evaluation and Updates

As previously mentioned, many corruption cases turn out to be caused by fault disk systems. Yet for as many of these I've seen, I also find that many customers don't keep their disk hardware and system up-to-date. Some disk vendors provide basic diagnostics you can run regularly for the disk system, and many have updates for their drivers and firmware (much like Microsoft has for its software). So, include evaluation and updates for your disk system (plus all of your server hardware) as a regular part of your maintenance plan and strategy.

Install the Windows PAE Fix (KB 838765)

If you read anything in this section, pay attention to what I'm about to tell you. Any Windows 2000 or 2003 server using the physical addressing extensions (PAE) should make sure they have the necessary Windows fixes as described in Microsoft Knowledge Base article 8387765 applied. If you are running Windows 2003 Service Pack 1, you are covered. If not, read the article and get the fixes the article describes.

This problem in Windows can result in unpredictable behavior, including access violations and database corruption. The reason is that the bug can result in unexpected frames from Windows pages for other processes to be written on top of memory pages for the SQL Server process.

Note that on some systems, PAE is enabled even if you don't specify the `/PAE` switch in your `boot.ini`. My advice is to get the fixes for this problem installed on your server.

Run *CHKDSK.EXE*

This program is often forgotten when analyzing root cause of corruption problems. If the `CHKDSK.EXE` program from Windows shows damage to the NTFS file system for the drive where SQL Server database and log files exist, your search for a cause should stop there. Any damage to the file system could result in database or log corruption.

One common question for `CHKDSK.EXE` is "Can I run the repair option for SQL Server files?" The answer is yes, but you lose data if repair moves NTFS clusters where SQL Server data is stored. The actions to take before running `CHKDSK` repair depend on your available backups and the state of the disk system. If you believe the disk system is not a problem (perhaps you don't see any disk-related problems aside from NTFS errors in the event log) and you can restore from a backup, proceed with `CHKDSK` repair. If you cannot restore from a backup, however, you might want to consider copying data from the SQL database in question before you do anything else. This is because `CHKDSK` repair could move clusters of your SQL Server database file, and you might not know what is really damaged or available to copy. One thing you should not do is rely on `DBCC CHECKDB` repair when NTFS problems exist. This is the same advice I have provided for disk system problems. If the disk is damaged, running `CHECKDB` repair is not a wise decision. Why repair SQL Server pages based on a faulty disk system? Who knows whether repair will work or itself encounter corruption problems.

Evaluate *CHECKDB* Results

Saving the result of DBCC CHECKDB when errors are detected is the most important piece of information to analyze the possible root cause of corruption. If you don't know what errors were encountered, it is difficult to discuss the cause of corruption. My philosophy is that you should save *any* execution of DBCC CHECKDB whether it was successful or not. Keep the results of CHECKDB in the same LOG directory where ERRORLOG files are kept.

Inspect Damaged Databases

In some cases, just looking at the errors raised by DBCC CHECKDB is not enough to understand a possible root cause. It is important to look at the actual damage to a page to understand a possible cause. It may be important to also look at previous copies of the database (through older backups) to look for any special patterns of the damage.

This is one reason why it is important to back up a database before you repair it or at least dump the pages with DBCC PAGE.

Let's say you encounter a checksum error on a page like the following:

```
Msg 824, Level 24, State 2, Line 1
SQL Server detected a logical consistency-based I/O error: incorrect checksum
(expected: 0xc3e99060; actual: 0x78dc1f61). It occurred during a read of page
(1:152) in database ID 17 at offset 0x00000000130000 in file 'C:\Program
Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\test.mdf'.  Additional messages
in the SQL Server error log or system event log may provide more detail. This
is a severe error condition that threatens database integrity and must be
corrected immediately. Complete a full database consistency check (DBCC
CHECKDB). This error can be caused by many factors; for more information, see
SQL Server Books Online.
```

Let's say you look at this page with DBCC PAGE and get the following output (I included only a portion of the page):

```
DBCC PAGE(17, 1, 152, 2)

DBCC execution completed. If DBCC printed error messages, contact your system
administrator.

PAGE: (1:152)


BUFFER:


BUF @0x02BEA7C0

bpage = 0x04DC0000                     bhash = 0x00000000
bpageno = (1:152)
bdbid = 17                              breferences = 3
bUse1 = 25322
bstat = 0xc00809                       blog = 0x999a2159
bnext = 0x00000000
```

```
PAGE HEADER:


Page @0x04DC0000

m_pageId = (1:152)                      m_headerVersion = 1
m_type = 1
m_typeFlagBits = 0x4                    m_level = 0
m_flagBits = 0x8200
m_objId (AllocUnitId.idObj) = 67      m_indexId (AllocUnitId.idInd) = 256
Metadata: AllocUnitId = 72057594042318848
Metadata: PartitionId = 72057594038321152
Metadata: IndexId = 0
Metadata: ObjectId = 2073058421       m_prevPage = (0:0)
m_nextPage = (0:0)
pminlen = 12                            m_slotCnt = 1
m_freeCnt = 8079
m_freeData = 111                        m_reservedCnt = 0
m_lsn = (37:79:3)
m_xactReserved = 0                      m_xdesId = (0:0)
m_ghostRecCnt = 0
m_tornBits = -1008103328

Allocation Status

GAM (1:2) = ALLOCATED                   SGAM (1:3) = ALLOCATED
PFS (1:1) = 0x61 MIXED_EXT ALLOCATED  50_PCT_FULL
DIFF (1:6) = CHANGED
ML (1:7) = NOT MIN_LOGGED

DATA:


Memory Dump @0x4454C000

4454C000:   01010400 00820001 00000000 00000c00 †................
4454C010:   00000000 00000100 43000000 8f1f6f00 †........C.....o.
4454C020:   98000000 01000000 25000000 4f000000 †........%...O...
4454C030:   03000000 00000000 00000000 6090e9c3 †............`...
4454C040:   00000000 00000000 00000000 00000000 †................
4454C050:   00000000 00000000 00000000 00000000 †................
4454C060:   10000c00 01000000 01000000 0200fc00 †................
4454C070:   68656c6c 6f776f72 6c640000 00000000 †helloworld......
4454C080:   0000ba03 00000000 00000000 00000000 †................
4454C090:   00000000 00000000 00000000 00000000 †................
4454C0A0:   c0000080 00000000 00000000 00000000 †................
4454D9A0:   00000000 00000000 00000000 00000000 †................
4454D9B0:   00000000 00000000 00000000 00000000 †................
4454D9C0:   00000000 00000000 00000000 00000000 †................
4454D9D0:   00000000 00000000 00000000 00000000 †................
4454D9E0:   00000000 00000000 00000000 00000000 †................
```

Let's say you know that this page is for a table that shouldn't contain any character columns. Notice the "helloworld" string stored on the page. This may be a hint to you that *something* wrote incorrectly the string helloworld on a database page in memory and then the page was written to disk. Now it is damaged. I've seen this exact type of problem occur when a buggy extended stored procedure wrote strings on database pages, causing corruption.

Use the Database/Log "Replay" Technique

Suppose you don't see any common disk system type of errors in the event log, `SQLIOSTRESS` doesn't encounter any errors, and `CHKDSK` doesn't report any problems. Now what? Well, first you should know that the underlying cause could still be the disk system. I've seen situations where the event log has no errors, `CHDDSK` is clean, and `SQLIOSTRESS` running for hours showed no errors. Yet when the customer replaced the SCSI adapter or disk drive, the problem went away. I don't have scientific explanations for these situations except to say that some problems could be timing-related and specific to IO patterns from the use of SQL Server in production. Existing tools or utilities don't expose the problem.

Having said that, I've also been in situations in which I suspected SQL Server could be the cause due to symptoms such as the following:

- The errors from `CHECKDB` are always with the same table and perhaps are even the same type of errors (say row damage to an index).
- A "replay" of a clean database full backup and a series of transaction log backups show the errors from CHECKDB.

I've seen situations where the first point is true, but the problem was still system-driven just because the index or table was a hotspot and was used frequently. I've even seen the second point being a system problem just because of the nature of what was recorded in the transaction log.

Having said that, if you can take a full database backup and restore a series of transaction log backups to replay errors from `CHECKDB`, you should engage Microsoft PSS. Be careful here. You must first ensure that the full database backup is clean. In other words, if you just restored the database backup, `CHECKDB` should report no errors.

The reason this technique is important is that it indicates some transaction log record can be redone or undone and cause an error to occur with `CHECKDB`. If you replay a backup and transaction logs and you don't find any errors from `CHECKDB`, the problem must have occurred on the original database or log file and is not *baked* into a record from the transaction log.

One possible explanation at this point is that a database page was damaged when it was written to disk or after being written to disk. Database checksum is the primary technique to discover if a page was damaged after SQL Server wrote it to disk. Let's look at another alternative and methods to detect damage to a page *before* it is written to disk.

Use SQL Server IO Audits

I've described in this chapter the new database checksum feature for SQL Server 2005. Now I show you some more advanced techniques used to detect certain types of damage to pages in addition to checksum. It is important to know that the use of some of these features, usually by trace flags, can have performance implications for your server. Database checksum was designed into the product and tested for performance (which is why it is the default option for a database in 2005). My recommendation is that you consult with Microsoft PSS when considering these options.

Stale Read (Trace Flag 818)

A *stale read* occurs when SQL Server writes a modified page to disk but the disk system returns a previous version of the page (perhaps from hardware cache). A *lost write* occurs when SQL Server modifies a page and writes it to disk, but the disk system never stores this modification to disk, so the previous version is returned when reading the page from disk. These differences are subtle, and the symptoms of the problem can appear to be the same: A modification to disk appears to be lost. One primary difference is that a stale read can result in correct data after something like a system reboot occurs to clear the cache.

It is important to know that this situation would not be detected by a database checksum. This is because the page itself is valid based on the checksum value; the hardware is simply giving back an unexpected version of the page.

In SQL Server 2000, the SQL Server development team added logic to detect this problem if an error for a page was encountered such as `Msg 823` or `605`. If you enabled trace flag 818, the engine would keep track of the latest LSN value for a database page in memory. When an error was encountered, the engine would compare the LSN value on the read page to the last known modified LSN in its "list of LSN values." If they didn't match, an error like the following was written to the `ERRORLOG`:

```
SQL Server has detected an unreported OS/hardware level read or write
problem on Page (1:75007) of database 12
LSN returned (63361:16876:181), LSN expected (63361:16876:500)
Contact the hardware vendor and consider disabling caching mechanisms to
correct the problem
```

In SQL Server 2000 Service Pack 4 and SQL Server 2005, the SQL Server development team enhanced the design of this trace flag to perform the LSN check on every read of a page and to store the LSN list in a more efficient hash table design. This auditing is not only by default and requires the trace flag to be enabled at server startup time. Again, you should use this trace flag only if you are having difficulty tracking down the cause of the corruption problem.

Eliminate Possible Memory Scribbler Software

If you think the problem is a damaged database page before it is written to disk, seriously consider eliminating any software running in the SQL Server process space that could cause unexpected memory damage. This includes user-written extended stored procedures, COM objects loaded by `sp_OA`, and linked server providers. I'm not saying this because it is a safe thing to do. I'm telling you this because I've seen things like user-written extended stored procedures have bugs that overwrite SQL Server database pages leading to corruption as they are written to disk.

If you can't disable these custom objects and procedures, considering moving them out of the process space for SQL Server. The SQL Server documentation provides information on how to run COM objects out of process with `sp_OA` and linked servers out of process when configuring the linked server. An extended procedure would be run out of process on the primary SQL Server by installing it on another SQL Server and executing it from the primary as a SQL remote stored procedure call.

Relocate Files to an Alternative Location

The last item on the root cause checklist may sound too simple, but I put this here because I've seen it work before. If you have tried all these options and are still scratching your head for a possible cause, it could still be your disk system. I've had customers simply move their database (or a system database such as `tempdb`) to another disk drive or disk system (such as a SAN or off of a SAN), and the problem has disappeared. To be fair and accurate, the move of the files might have changed the timing of the problem, and it could still be SQL Server. In many of these cases, however, I've seen customers then replace the original disk controller or disk system, and the problem will permanently go away. Don't discount this simple technique when you are thinking of all of these trace flags and tools to use.