

Database in SUSPECT state, what now?

Know the probable causes and solutions for this problem

One of the worst situations that a Database Administrator can face is to hear complaints from all users of the system reporting that they cannot access the database or that the application has access errors in the database. After a brief check of this situation, the DBA discovers that one of the production databases is inaccessible and has entered the *SUSPECT* state.

To exemplify how to work in this situation, we will use two practical cases. A corrupted log file and how to put the database *ONLINE* without using the backup restore feature, assuming that this feature is not available at this moment. With the other case, we will work with the hypothesis of the log file has been accidentally deleted while the SQL Server service was down or even if the log file was irretrievably corrupted, and will require an action without counting with this file. To carry out these examples and guide the theory, the version of SQL Server used here must be 2005 or later.

In SQL Server, we have “database states” that are representations of how the database is found for process executions. The states define whether the database is able to work on it or if there is a problem that needs direct action by the DBA. These states are shown below:

- **ONLINE:** The database is available for access. Here they can access and record information;
- **OFFLINE:** The database is unavailable. The SQL Server service does not automatically put the database in this state. It is a state that only the database administrator places it via the ALTER DATABASE [database] OFFLINE command;
- **RESTORING:** The database is unavailable. One or more files in the filegroup that make up the database are being restored by a restore action;
- **RECOVERING:** The database is being recovered. It will be available once the recovery is complete. If you cannot recover (put the database ONLINE) it goes into SUSPECT state and is inaccessible. The “RECOVERING” state happens every time the SQL Server service is started to test that all objects and their data files are healthy;
- **RECOVERY_PENDING:** The database is will to some error recovery and further action is required from the user to resolve the error and allow the recovery process to complete. The database is inaccessible as long as this error is not remedied;
- **SUSPECT:** Some of the files in the database group are corrupted. When SQL Server starts, it checks that the files are healthy. If he finds a problem in these files he puts the database in the “SUSPECT” state. This state requires the direct action of the Database Administrator and is the state that we will work on in this article;
- **EMERGENCY:** In this state the database is in single user mode and marked as READ_ONLY in addition to the recording log file of the database is disabled. It is in this state that we will work to resolve the state of the SUSPECT database and to make it ONLINE.

The database in the “ONLINE” state is accessible and ready to be connected, with all data recording (such as update, insert or delete) going through the log file and then going to the datafile file.

In this article we will work with databases in the SUSPECT state. The SUSPECT state means that there is a transactional and / or structural inconsistency. This is the result of a failed recovery process

(typical when data page corruption occurs) or the SQL Server service started, but failed to RECOVER the database (typical when an error occurred in the database log or also corruption in data pages).

Because of this unsuccessful attempt to put the database online, SQL Server cannot guarantee data consistency and marks it as SUSPECT. In addition, the database will be found as if it were offline (inaccessible), and the only possible action on that database is to go to the “EMERGENCY” state (read only).

Reasons to corrupt a database

Most I/O problems occur due to subsystem failures, such as failure of the operating system or storage drivers with firmware problems at some point. In these cases, when writing data from RAM memory (or from the SWAP file) to disk, a failure can occur where a page, a header or even the entire database file can become corrupted. Even a sudden power failure at the time of recording can also corrupt the database.

When we download the SQL Server Service abruptly (with the SHUTDOWN WITH NOWAIT command), the SQL Server service can write incomplete information to storage and can also cause corruption, as it does not expect the actual data to be written to disk.

Another important detail that can corrupt the database is to leave the AUTO CLOSE option to true. This means that, as soon as the last session ends within the database, the physical files that make up the database (*.mdf and *.Ldf) are free for other operating system services to use.

With the AUTO CLOSE option set to true, compressing Windows files or even moving one or all of the database's datafiles is entirely possible. Even the antivirus scan itself can block the use of datafiles in SQL Server and not leave the database ONLINE.

That is why it is important to create a rule in the antivirus application so as not to check the SQL Server datafiles folder. This prevents the antivirus from attempting to access these files under any circumstances and also frees any competition between the SQL Server service and the antivirus.

Another situation that can end up corrupting a database datafile is to leave the option “Page Verify” to NONE or TORN_PAGE_DETECTION. The NONE option does not use the verification of data recording on the disk and the other option (TORN_PAGE_DETECTION) is already outdated in versions SQL Server 2005 or higher (since it only uses 2 verification bytes in the header of the data page). Always leave it in CHECKSUM, because it uses a hash in the header of the data page, much more secure and accurate than the TORN_PAGE_DETECTION method. With SQL Server using CHECKSUM to guarantee data entry on disk, it will not encounter an error when trying to recover the data. Otherwise (NONE or TORN_PAGE_DETECTION) makes the recording without the precise confirmation of the data, causing errors to be issued (as in a select) and data will be lost. It is good to remember that there is no performance degradation in leaving the “Page Verify” option in CHECKSUM.

Upgrading versions of SQL Server can corrupt datafiles (yes, database datafiles can corrupt!). So always back up your databases before performing a version upgrade (cumulative patches also follow the same rule).

Even disk defragmentation can cause datafiles corruption, according to the Microsoft article (A heavily fragmented file in an NTFS volume may not grow beyond a certain size) in the links section. This article indicates that for very large files I/O failure and corruption may occur.

Another reason for putting the database in “SUSPECT” state is modifications in the powers of the account that starts the SQL Server service. These modifications can make the SQL Server file datafiles inaccessible, resulting in an apparent, corrupted database. However, in this case, just fix the access and restart the SQL Server service, which returns everything to normal, without having to apply any correction method.

Corruption error messages

Below, some error messages that SQL Server provides when the data is showing corrupt behavior and it is necessary to check which problem caused these alerts:

- Msg 823 (error message) in SQL Server;
- Msg 825 and 824 (read retry) in SQL Server;
- SQL Server Page Level Corruption;
- SQL Server Table Corruption Error;
- Corruption error in Metadata (data from SQL Server system tables);
- Errors reported by the DBCC CHECKB command;
- Corruption in non- clustered indexes.

Above we find some corruption error messages, but there are thousands of others and for each one the main recipe is to stay calm, checking what is the source of the database problem!

Main features of a corrupted database

Whenever you hear of any of these symptoms below, be alert, as your database may be showing corrupt database behavior. For this reason, it is always important, in the daily checklist, to go through all the logs that the Windows servers and SQL Server provide:

- Users or application reporting I/O errors 823 and 824 (hardware and software, respectively);
- SQL Server alerts agent warning;
- SQL Server alerts with error severity codes greater than or equal to 19;
- SQL Server alerts with severity codes with type 825 errors (momentary I/O problems);
- Failure in backup plans, presenting error 3043 (backup detected “CHECKSUM ERRORS”);
- SQL Server Logs with recurring I/O failure errors, like this:
“SQL Server has encountered X occurrence (s) of I/O requests taking longer than XX seconds to complete on file [PATH] in database [Bank] (XX). The OS file handle is 0x00000000000000XXX. The offset of the latest long I/O is: 0x00000XXXaX0000 ”.

This error is about storage failure where the datafiles are located.

- Application / user connection fails, having the user try to connect to the system again;
- The Windows event viewer reports errors related to I/O in the operating system.

Steps to start scanning a suspicious database for corruption

So. There is a database that has some of the above characteristics or another that indicates an I / O problem. What to do? First, don't panic. At these times it is important to remain calm and work focused on solving the problem.

Determine the extent of corruption by running the DBCC CHECKDB command on the database with possible data corruption. The mentioned command will check the physical and logical integrity of the objects found within the database, scanning the entire database for flaws. This verification works at six levels, namely:

- Database disk space allocation structure;
- Integrity of all pages and structures of the database tables;
- Consistency of the database data catalog;
- Indexed content of the database;
- Metadata links to SQL Server system tables;
- Service Broker validation of the database.

It is worth mentioning that tables that have activated memory optimization (feature available since version 2014), the mentioned command will not perform the repair, only checking for possible errors.

Note: Always wait for the DBCC CHECKDB command to run, otherwise it will actually corrupt the database. Keep in mind that the bigger the bank, the longer it will take to finish the command.

- Look at the alerts and the SQL Server log and see if the problem still persists by determining how long it has been happening. Some read and write failures are automatically corrected by the operating system, causing SQL Server to save the new information elsewhere. In addition, your database may have been corrupted for so long that the backups that exist are also with the corrupted databases stored in the backups. The backup is intact and it is possible to perform the restore, but the data contained in that backup is corrupted;

- If there are Jobs that make backups, check if they are in error. It is important to note that backup jobs have alerts to warn if there is an error in execution. Check that the location where the backups are stored has enough space for the time window programmed by the organization;

- See if backups are available and test restores regularly before having the “surprise” of the database showing suspect status. The most important and safest method is to restore the last valid database backup;

- Make backups of logs of the databases that have the option of Recovery model in “FULL”. With the log backups up to date it is possible to restore the database at any time in time (POINT IN TIME) and also make tests on these files. Many companies fail this test and if any of the log backup files are corrupted it compromises the entire restore chain, forcing the restore only from the point of the FULL backup.

And how to interpret, when executing the command DBCC CHECKDB on the server, fatal errors that require the application of a backup restore to the database or the reconstruction of the database from its structure?

Prior to any action, if the DBCC CHECKDB command fails before it ends, some very serious problem must be occurring in the bank. This means that there is no option but to trigger a restore and return a valid backup.

Some fatal error codes when running the DBCC CHECKDB command. These codes represent a serious problem with your database and need immediate action by the administrator, checking in which table or structure the error occurred:

- 7984 to 7988: Critical corruption in the system tables;
- 8967: Invalid state of CHECKDB itself;
- 8930: Corrupted metadata. CHECKDB is unable to correct this type of error as it is a corruption of SQL Server metadata. Metadata, in this context, is the SQL Server system catalog.

The errors below are also pointed out by the DBCC CHECKDB command and it can not be solved (but possibly has a solution, using other tools or other solutions that SQL Server provides):

- 8909, 8938, 8939: (Corrupted data page header);
- 8970: Invalid data for the column type. Some type of data was found as a special character (Ex. ‘Ð’) in a column that doesn’t accept that type of data;
- 8992 error: CHECKCATALOG (metadata incompatibility). This problem occurs because SQL Server does not support manual updates to system tables. System tables should be updated only by the SQL Server database engine;
- 8904: Size for allocating two objects. For some unexpected disk failure, SQL Server writes the space required for one object as being for two objects, taking up unnecessary space and making data difficult to read.

Tips for analyzing a database

We will now see some tips and tricks to check what action we should take when a database shows corrupt behavior. We will know if it is faster (and less laborious) to use a database restore or try to recover the data using the DBCC CHECKDB command:

- Do you still have the online database? If not, try searching for the solution from a backup. Remember that recovery using backups is not the best way to avoid downtime as it will depend on the backups being accessible at the moment and the size of the databases involved for the restore, but it is the best way to recover most (or all) of the data. Remember: Back up your databases and always test your backups in another environment to avoid surprises in the future;
- Did the DBCC CHECKDB command fail? If so, you must seek recovery in a restore, necessarily;
- If the error occurred on non-clustered indexes, (corrupted index), then you may be able to use the ALTER INDEX ... REBUILD option (never ONLINE) to correct the problem. There will be performance degradation at the time of recreating the index.
- Do you have backups working? If not, try to restore with CONTINUE_AFTER_ERROR, or extract the data to a new database. After that, copy it later to the production database.
- When the database log is corrupted, you can restore a log backup or perform the emergency recovery state, which will be applied in the examples that we will see later.

How to repair a database ?

After checking the database we decided to repair it with possible data loss. In this case, the database will be available for access after correction, but we will have data that cannot be saved. Below we will see how we can carry out this repair:

Using the command DBCC CHECKDB ([database], REPAIR_ALLOW_DATA_LOSS) with the parameter REPAIR_ALLOW_DATA_LOSS we repair the fixed structures of the data pages, relocating the data to another valid area. In addition, it can also perform index reconstruction. The repair will not take into account (will not correct or restructure) foreign keys constraints, inherent business logic and data relationship and SQL Server replications.

Before executing the command to repair, protect yourself by making a backup of the database in its current state and stop replication, if it exists. After performing the recovery, check the data again by re-executing the DBCC CHECKDB command (without parameters) and activate again any replication involved (if any) in addition to performing a new backup of the database.

Database log is damaged

While the system was operating, for some reason the production database presented one of the corruption problems mentioned above. We took a look at the state of the database and it is corrupted. We will now see what can be done.

First, check what type of error is occurring in the log backup. Also take advantage and check the SQL Server logs and qualify the error (understand what type of error SQL Server is presenting). Try to recover the log from a recently made log backup, as this will always be the best choice.

Put the bank back in the ONLINE state and work on correcting the source of the problem (if it is storage, driver, service pack, etc.) and do not forget to schedule a maintenance window to correct this problem and pay attention to the error notices of the SQL Server.

When checking to return a backup, it was found that there is no backup available. In this case, we have two realities. The first is to use the state of the EMERGENCY database to be able to access the database and work on it. To do this, use the command ALTER DATABASE [bank] SET EMERGENCY. In addition, the database must be in SINGLE_USER mode;

In the EMERGENCY state, we can extract the data and write to another non-corrupted database, reconstructing the database manually. This reality is the most difficult.

The second is to use the DBCC CHECKDB command with the REPAIR_ALLOW_DATA_LOSS option to reconstruct the log file (if possible). Remembering that in this case we will possibly have data loss.

What not to do? What are the worst practices?

Below are some examples of actions that we normally take in a critical environment that can create even more problems, instead of solving them. These items cover practices that are very common for unsuspecting DBAs:

- Restart the SQL Server service. When we restart the SQL Server service it tries to put all banks in the “ONLINE” state and checks each state of the databases' datafiles. It will only put the bank in a “SUSPECT” state and will not solve the problem. Restarting the SQL service becomes a waste of time, in addition to taking other databases from the same instance down;
- Skip the steps to check the type of error and apply a database restore without actually checking the need (trying to recover the data or reconstruct the logs);
- Detach (“detach”) a bank in the “SUSPECT” state. It will surely fail when you attach again. This is the worst situation and should be avoided as much as possible! In this case, the log file will surely lose data and all transactions that were in it will be lost. There is a possibility to solve this case using a hack method (“jaqued-up”) which will be shown later in the article.

Example 1 - Database with corrupted log

First, we created a database called BaseComLogSuspeito that will be used to create our model structure and also to insert the data and then, purposely create the corruption in the database. This database will be created in SQL Server default mode without any custom parameters. Use SQL Server Management Studio to complete the following steps and listings. Within the BaseComLogSuspeito bank, we will create a table with information from student grades.

Note: It is important to remember that to perform this example, we will be restarting the SQL Server service several times. Therefore, use an environment where it has no impact on carrying out this process. In this example, we will create a table (with the name NotaDosAlunos) not following the rules of Normal Forms, that is, without respecting the normalization of data and its 5 normal forms. It will only serve the purpose stated.

Listing 1. Creating the database and populating

```
USE [master]
GO

CREATE DATABASE [BaseComLogSuspeito]
GO

USE [BaseComLogSuspeito]
GO

CREATE TABLE [NotaDosAlunos]
(
    [NomeDoAluno] VARCHAR (50),
```

```

        [Turma]                VARCHAR (30),
        [Nota]                  float
    );
GO

INSERT INTO [NotaDosAlunos] VALUES ('Eduardo', 'A', 7.8);
INSERT INTO [NotaDosAlunos] VALUES ('João', 'B', 10);
GO

```

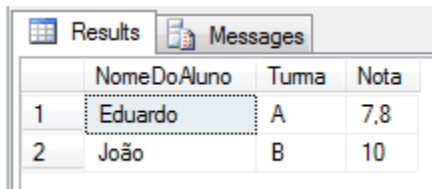
Let's check if the data was successfully saved in the table, through Listing 2 and the result in Figure 1. These data will be used for the purpose of recording the information in the log file and, afterwards, corrupt them in order to be able to recover them later:

Listing 2. Checking if the data was successfully saved

```

SELECT      [NomeDoAluno],
            [Turma],
            [Nota]
FROM [NotaDosAlunos]

```



	NomeDoAluno	Turma	Nota
1	Eduardo	A	7.8
2	João	B	10

Figure 1. Result of data inclusion

We have successfully recorded two records in the NotaDosAlunos table. Now we are going to accidentally update Eduardo's note by placing the value zero. Thus, we will force the update to be written to disk, using the CHECKPOINT command, as shown in Listing 3, ensuring that the information is written to the log files.

Listing 3. Performing a transaction by forcing its recording with *CHECKPOINT*

```

BEGIN TRAN;
UPDATE
    [NotaDosAlunos]
SET
    [Nota] = 0
WHERE
    [NomeDoAluno] = 'Eduardo';
GO

CHECKPOINT
GO

```

In another window, we will simulate a power outage by downloading the SQL Server service abruptly with the WITH NOWAIT parameter (Listing 4). The NOWAIT parameter shuts down the SQL Server service without performing checkpoints on the entire database:

Listing 4. Stop the SQL Server service abruptly

SHUTDOWN WITH NOWAIT
GO

With the SQL Server service stopped, we will simulate a physical damage in the log file, using the HxD software (session links) which is a FREE memory editor and Hexadecimal file.

To do this, first install the software (next, next, finish). After installation, run the software and in the file menu, open option and go to where the log file is. Load it and a screen like Figure 2 will appear.

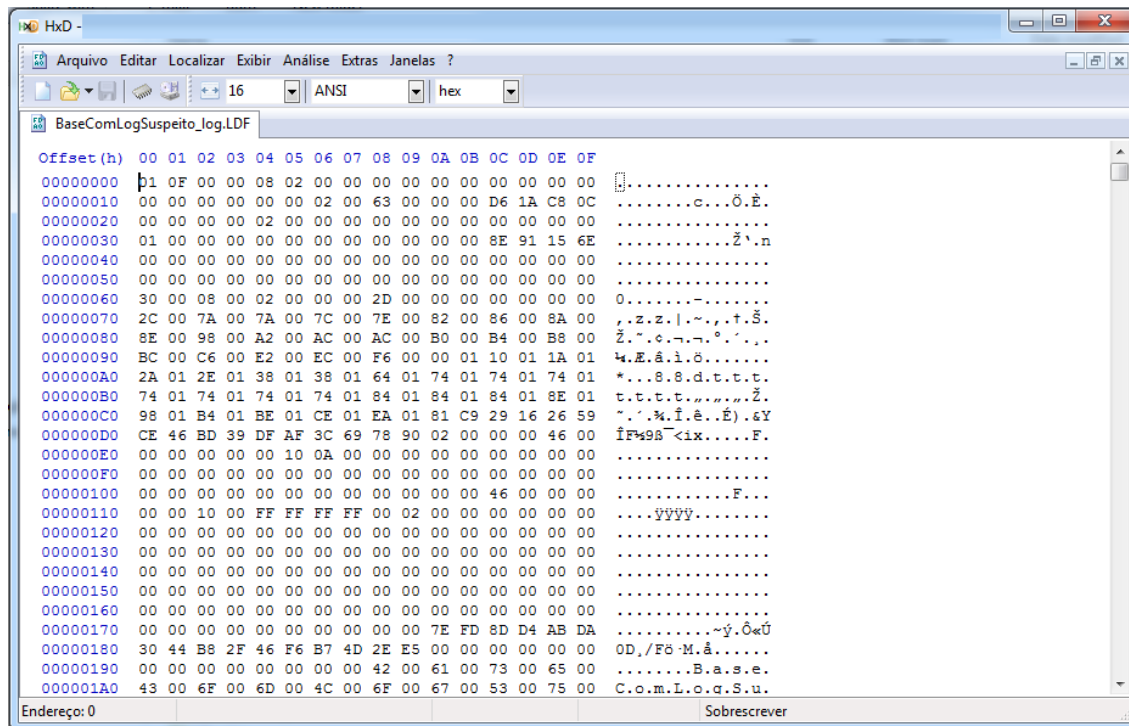


Figure 2. Hexadecimal Editor screen

In the first line where 00000000 is located (below Offset (h)) fill the line with zeros (“0”) until the end of it, to simulate a log file recording error and leave it corrupted, as shown in Figure 3:

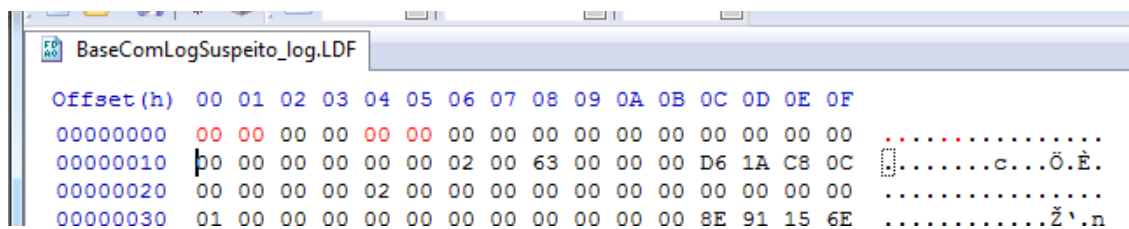


Figure 3. Line to edit and simulate a log file corruption error.

Save the file in the same folder as the original file (replacing the file). Note that the software itself already creates a backup file, called BaseComLogSuspeito_log.LDF.bak without having to interact and create a backup file. If any problem occurs and you cannot simulate the error, delete the file edited by the HxD software and rename the backup to it's original name.

Now we are going to put the SQL Server service back online. In the Command Prompt window of the server (cmd.exe), we execute the command: NET START MSSQLSERVER (assuming the SQL Server instance is the default). Follow in Figure 4.

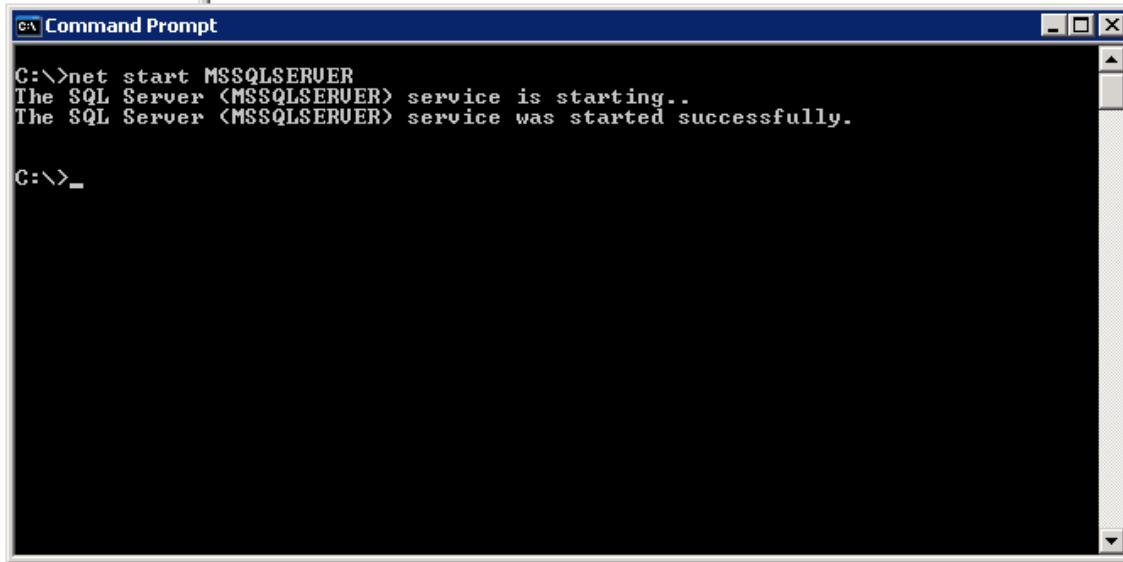


Figure 4. **Figure 4. Starting the SQL Server service**

With the SQL Server service on the air, we tried to enter the database (Listing 5), where it will present the error in Figure 5, warning that the BaseComLogSuspeito database does not have its files accessible or there is not enough memory on disk.

Listing 5. Accessing the database

```
USE [BaseComLogSuspeito]
GO
```

We are faced with the following message (Figure 5):

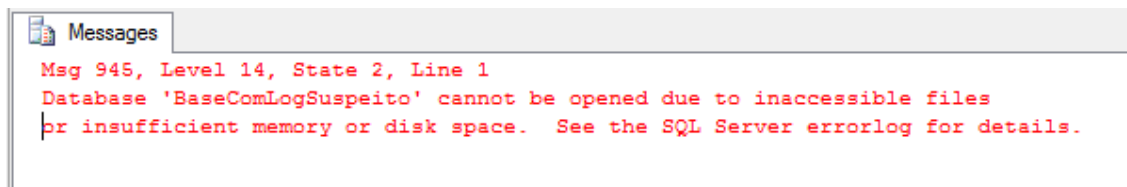


Figure 5. Error screen due to the log is corrupted

Through Listing 6, we verify the state of the database. In this case, the database will be in “SUSPECT” state, making it impossible to carry out transactions on it. In this state, the database only allows changing its state to “EMERGENCY, a state in which data can only be read in the database.

Listing 6. Checking the database state

```
SELECT DATABASEPROPERTYEX (N'BaseComLogSuspeito', N'STATUS') AS N'Status';
GO
```

And it is in “*SUSPECT*” state, as shown in the screen below, in **Figure 6**:

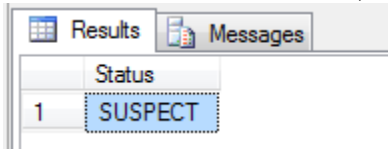


Figure 6. Database status in “*SUSPECT*”

Our database is already in the state we wanted to be able to run the laboratory. So the first step is to put the bank in an “EMERGENCY” state (read-only) and in SINGLE_USER mode (only one user accessing the database), according to Listing 7, in order to execute the DBCC CHECKDB command. It is important that if you do not put the database in SINGLE_USER mode, it will not let you perform any repairs on the database.

Listing 7. Fix the state of the database

```
ALTER DATABASE [BaseComLogSuspeito] SET EMERGENCY;
GO
ALTER DATABASE [BaseComLogSuspeito] SET SINGLE_USER;
GO

DBCC CHECKDB (N'BaseComLogSuspeito', REPAIR_ALLOW_DATA_LOSS) WITH NO_INFOMSGS,
ALL_ERRORMSG
GO
```

We run the command DBCC CHECKDB with the options NO_INFOMSGS (do not display informational messages) and ALL_ERRORMSGS (display all error messages). The REPAIR_ALLOW_DATA_LOSS option, in this case, retrieves the log file and deletes the pages where the corruption is, recreating the same empty ones. The result is shown in Figure 7.

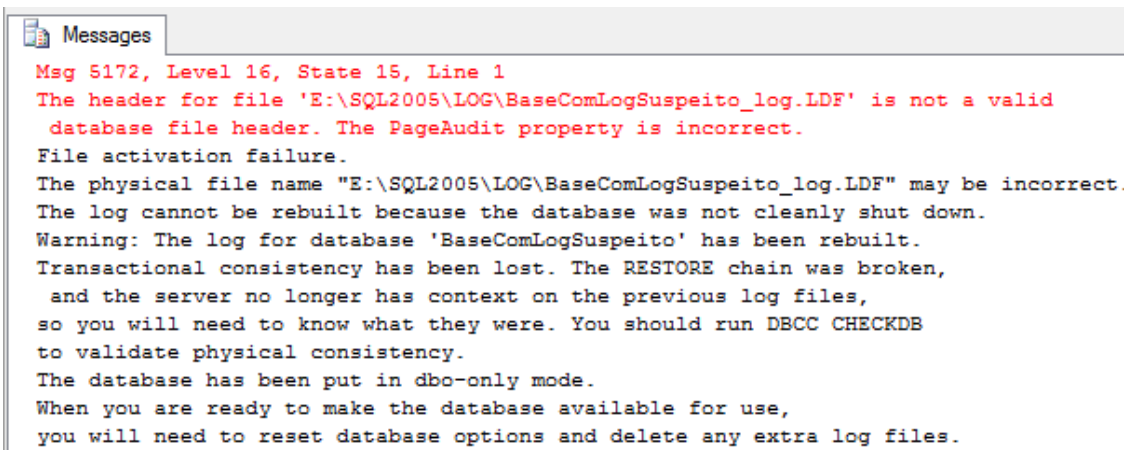


Figure 7. Database log recreated

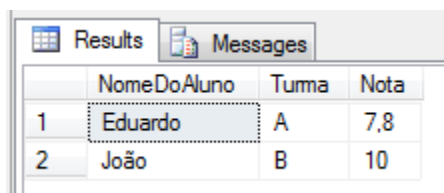
What the DBCC CHECKDB command did was to rebuild the log file (The log for database ‘BaseComLogSuspeito’ has been rebuilt). But were the data saved from that UPDATE that was performed?

We will try to enter the database again and see how the table [NotaDosAlunos] looks like. But first, it is necessary to put the database “ONLINE” again and in multi-user mode, as shown in Listing 8. The result of executing these commands is shown in Figure 8.

Listing 8. Putting the database back “ONLINE” and checking the data

```
ALTER DATABASE [BaseComLogSuspeito] SET ONLINE
GO
ALTER DATABASE [BaseComLogSuspeito] SET MULTI_USER
GO
USE [BaseComLogSuspeito]
GO

SELECT      [NomeDoAluno],
            [Turma],
            [Nota]
FROM [NotaDosAlunos]
GO
```



	NomeDoAluno	Turma	Nota
1	Eduardo	A	7,8
2	João	B	10

Figure 8. Recreated log base

Note that he did not record the information, as the command had not yet ended with COMMIT (recording to disk in the data file), but only recorded the information in the log file (CHECKPOINT).

What if this command was not accidental? Then we would have data loss, which would be expected for the execution of this command. Its advantage is that, depending on the size of the database and the restore process (searching for the restore file, restoring it and putting the database online again) it takes longer and it is more worth losing some data and entering the same data manually after the bank is “ONLINE”.

Example 2 - Database with deleted *log*

As in Example 1, we will create the base and create the table with the data (Listing 9). In this new example, we will untack the database and delete the log file from the database, creating a situation where it is impossible to recover the log file (assuming that the log file is not backed up). In Listing 10 we run the update to simulate a transaction and, in another window, we execute the command in Listing 11, shutting down the SQL Server service normally:

Listing 9. Creating the database and populating it

```
USE [master]
GO

CREATE DATABASE [BaseSemLog]
GO

USE [BaseSemLog]
GO

CREATE TABLE [NotaDosAlunos]
(
```

```

        [NomeDoAluno] VARCHAR (50),
        [Turma]        VARCHAR (30),
        [Nota]         float
    );
GO

INSERT INTO [NotaDosAlunos] VALUES ('Eduardo', 'A', 7.8);
INSERT INTO [NotaDosAlunos] VALUES ('João', 'B', 10);
GO

```

Listing 10. Creating the database and populating it

```

BEGIN TRAN;
UPDATE
    [NotaDosAlunos]
SET
    [Nota] = 0
WHERE
    [NomeDoAluno] = 'Eduardo';
GO

COMMIT
GO

```

Listing 11. Normally downloading the SQL Server service

```

SHUTDOWN
GO

```

We go to the folder where the SQL Server datafiles are located and delete the file BaseSemLog.ldf (database log file). After that, we will put the SQL Server service back online. In the Command Prompt window of the server (cmd.exe), we execute the command: NET START MSSQLSERVER (assuming the SQL Server instance is the default).

With the SQL Server service on the air, we try to enter the bank (Listing 12), noting that again we have the error (Figure 9) similar to Example 1 and we look at the state of the bank, checking that it is in “SUSPECT” mode (Listing 13 and Figure 10).

Listing 12. Access the database

```

USE [BaseSemLog]
GO

```

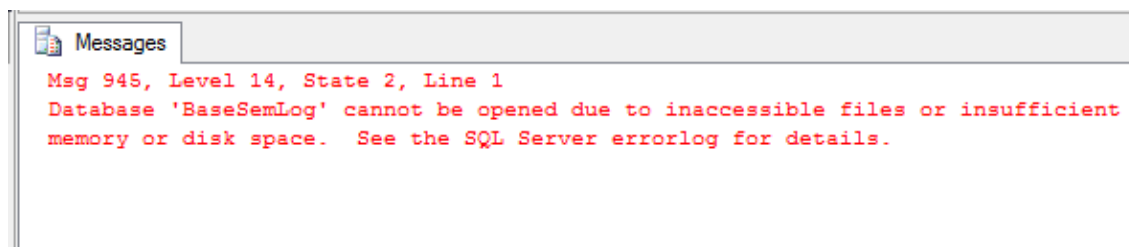


Figure 9. Again we have the error similar to example 1

Listing 13. Entering the database

```

SELECT DATABASEPROPERTYEX (N'BaseSemLog', N'STATUS') AS N'Status';
GO

```

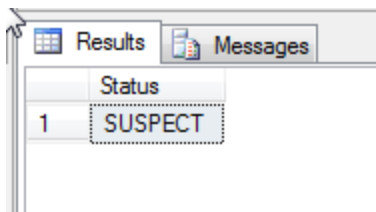


Figure 10. “**SUSPECT**” state

At this point, instead of trying to retrieve the datagabase with the DBCC CHECKDB command, let's try to untack the database (Listing 14), trying to put it right back on the server. In SQL Server 2008, the error in Figure 11 is shown, not allowing the bank to be detached, in SQL Server 2005 it is possible, presenting the message in Figure 12.

Listing 14. Trying to challenge Nexar the base 'BaseSemLog'

```
EXEC sp_detach_db 'BaseSemLog';
GO
```

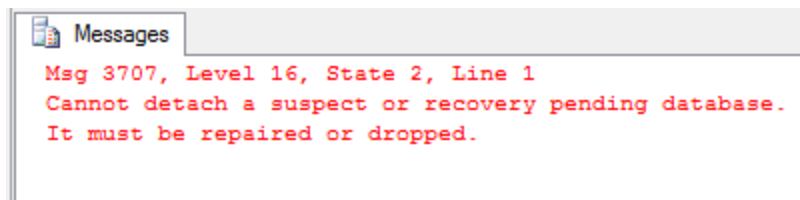


Figure 11. Error detaching 'BaseSemLog' in SQL Server 2008

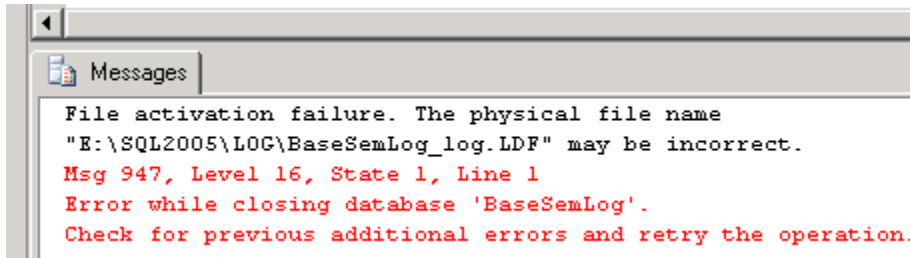


Figure 12. Detach message in SQL Server 2005

In the version of SQL Server 2005, he was able to detach the database, but he said he was unable to successfully shut down the database on the server. In this case, when “detach” occurs in version 2005, we can still put the database on the air through Listing 15, recreating the database a new log file (Figure 13).

Listing 15. Attach the base again without the log file in SQL Server 2005

```
EXEC sp_attach_single_file_db @dbname = 'BaseSemLog',
@physname = N'E:\SQL2005\DATA\BaseSemLog.mdf';
GO
```

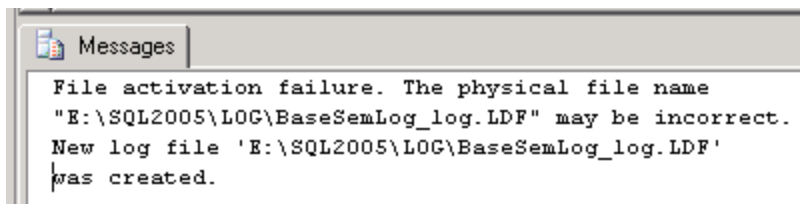


Figure 13. **Base attached with recreated *log* file**

To solve the problem of total loss of the log file in SQL Server 2008 (different from the 2005 version) we will use the `sp_resetstatus` command which disables the database state of “SUSPECT”, updating the `sys.databases` metadata of the database master. To illustrate this, we will execute the command in Listing 16 and the message will be displayed (Figure 14):

Listing 16. Resetting the database status

```
EXEC sp_resetstatus 'BaseSemLog'
GO
```

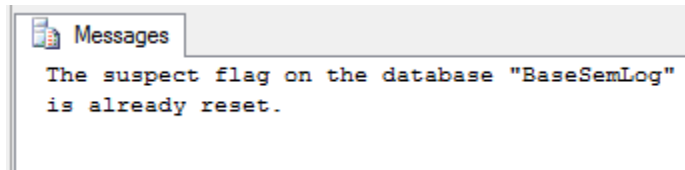


Figure 14. **Reset of 'BaseSemLog' base *status***

After this command it is possible to restructure the log base, as in example 1, with the commands in Listing 17, where the log file will be completely restructured, but without CHECKPOINT information. The result is shown in Figure 15, where it shows the message that it will be necessary to run a new DBCC CHECKDB to check the consistency of the data. After that, it is necessary to grant access to the multi-user option with the command below:

```
ALTER DATABASE BaseSemLog SET MULTI_USER
```

Listing 17. Restructure the log base

```
ALTER DATABASE BaseSemLog SET EMERGENCY
ALTER DATABASE BaseSemLog SET SINGLE_USER WITH ROLLBACK IMMEDIATE
DBCC CHECKDB ('BaseSemLog', REPAIR_ALLOW_DATA_LOSS) WITH NO_INFOMSGS
```

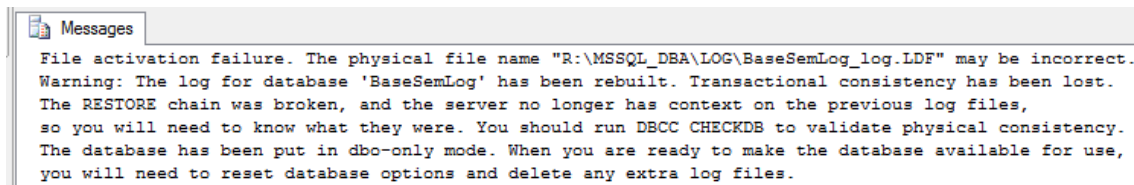


Figure 15. **Recreated 'BaseSemLog' database log file.**

Note that, in all of these cases, the log file has been reconstructed and you cannot save the information in it. In addition, there is the possibility of some inconsistency (again) in the database. So, run a DBCC CHECKDB immediately to see the “health” of the database.

With this action, a lot of information can be lost, information that was still inside the log file and was not transferred to the data file (*.mdf). This will also affect the performance of SQL Server, as by recreating the log file, it will be at its minimum size. And SQL Server uses the log file to work on retrieving the information in the database, having to redesign, relocate and open new physical space in the storage. Many problems related to deadlock are due to this. When recreating the log file or when executing a Shrinkfile on the log file with the option TRUNCATEONLY.

Box 1. Freeing up space with the TRUNCATEONLY option

Many indicate running the commands below in order to free up disk space:

```
ALTER DATABASE BancodeDados SET RECOVERY SIMPLE WITH NO_WAIT
GO
DBCC SHRINKFILE (N'BancodeDados', 0, TRUNCATEONLY)
GO
ALTER DATABASE BancodeDados SET RECOVERY FULL WITH NO_WAIT
GO
```

However, this case leaves the log file at its minimum size, forcing the information to be written to the data file, besides, all query optimization (used in cache) is lost and SQL Server is forced to restart its architectures. to optimize queries from scratch. Just like losing the log file and having to recreate it (as in example 2).

Therefore, make backups of the log files and leave it to the SQL Server itself to manage the free space of the log file.

Check with the DBCC OPENTRAN command which transactions are still active in the database and also check what they are doing, with the DBCC INPUTBUFFER (SPID) command, where SPID is the session found in the DBCC OPENTRAN command.

It is better to bring down a transaction only than to lose data or cause the entire system using the database to log reconstruction to be slow.

If even trying the commands in Listing 17 to reconstruct the log the bank does not return to the “ONLINE” state and continue as if it were corrupted, there is a trick to, in the last case, recover the database and put it back into production. It is called “jacked-up”.

First, we have a database in “SUSPECT” mode and we will also reconstruct the data log file, so we will only find the data file (*.mdf) in the SQL Server datafile folder. The log file can be deleted for this test. So we created the base and downloaded the SQL Server service with the commands in Listing 21.

Listing 21. Creating the database and stopping the SQL Server service

```
USE [master]
GO

CREATE DATABASE [BaseSemLog]
GO

USE [BaseSemLog]
GO

CREATE TABLE [NotaDosAlunos]
(
    [NomeDoAluno] VARCHAR (50),
    [Turma] VARCHAR (30),
```

```

        [Nota]                float
    );
GO

INSERT INTO [NotaDosAlunos] VALUES ('Eduardo', 'A', 7.8);
INSERT INTO [NotaDosAlunos] VALUES ('João', 'B', 10);
GO

SHUTDOWN
GO

```

We deleted the BaseSemLog_log.LDF file from the SQL Server datafiles folder and put SQL Server back on the air, with the net start MSSQLSERVER command, via the command prompt. In addition, we checked the state of the bank with Listing 22 with the result “SUSPECT”, as expected.

Listing 22. Creating the database and stopping the SQL Server service

```

SELECT DATABASEPROPERTYEX (N'BaseSemLog', N'STATUS') AS N'Status';
GO

```

Ok. The database is in the "SUSPECT" state and now we are going to apply the "jacked-up". To do this, disable the SQL Server service again, as done previously, rename the file BaseSemLog.mdf to BaseSemLog_OLD.mdf and put the SQL Server service on the air.

The database will still be presented as a “SUSPECT” state, but it only exists in the SQL Server metadata. In this case, we deleted and created the database again with the command in Listing 23. What will happen here is that it will only update the SQL Server metadata, since the BaseSemLog.mdf files for BaseSemLog_log.LDF no longer exist.

Listing 23. Creating the database and stopping the SQL Server service

```

DROP DATABASE BaseSemLog
USE [master]
GO

CREATE DATABASE [BaseSemLog]
GO

```

With the creation of the new database it will create the same data files as the previous situation (*.mdf and *.ldf). Then we stop the SQL Server service again, delete the newly created data file called BaseSemLog.mdf, rename the BaseSemLog_OLD.mdf file to BaseSemLog.mdf and put the SQL Server service on the air.

At this point, we will still find the database in the “SUSPECT” state. That's because again, the SQL Server metadata does not match the characteristics of the new data file BaseSemLog.mdf. To correct this, we use the commands in Listing 24 showing the result in Figure 16 where it relates the database to its metadata.

Listing 24. Creating the database and stopping the SQL Server service

```

ALTER DATABASE BaseSemLog SET EMERGENCY
ALTER DATABASE BaseSemLog SET SINGLE_USER
DBCC CHECKDB ('BaseSemLog', REPAIR_ALLOW_DATA_LOSS) WITH NO_INFOMSGS

```

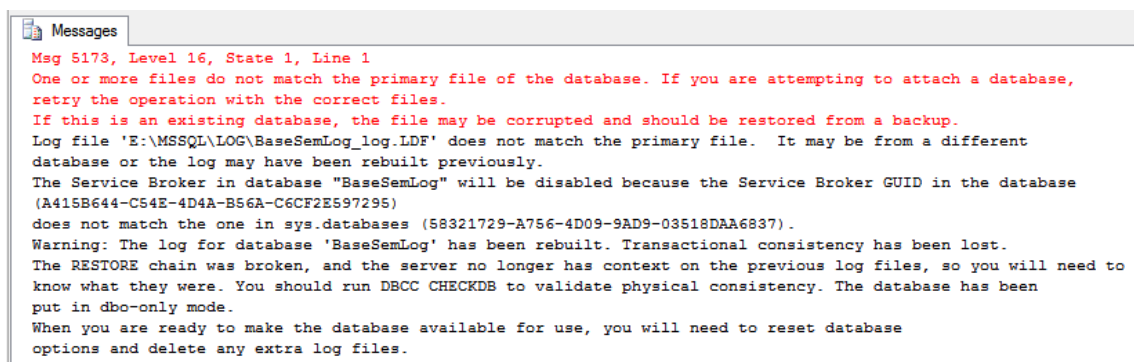



Figure 16. Database in “**SUSPECT**”.

Here he makes it clear that the GUID (the database's unique identifier in the metadata) are not the same. So he recreated the log file and also notice that the Service Broker service has been disabled for this database. However, the data was again released for access. To check, release multiuser access in the bank and check if the data are in the table (Listing 25 and Figure 17).

Also note in Figure 16 that he asks for the DBCC CHECKDB command to be executed again to check its consistency. Running this command again in the database will have a verification completed without errors, because now the database is, again, intact and ready to be released for production.

Listing 25. Creating the database and stoping the SQL Server service

```
ALTER DATABASE BaseSemLog SET MULTI_USER
USE [BaseComLogSuspeito]
GO
```

```
SELECT      [NomeDoAluno],
            [Turma],
            [Nota]
FROM [NotaDosAlunos]
GO
```

	NomeDoAluno	Turma	Nota
1	Eduardo	A	7,8
2	João	B	10

Figure 17. The data is accessible again in the database

Conclusion

We worked on this post with the possible causes for a database to enter the “SUSPECT” state and its respective solutions to put the database back online (“ONLINE”). We see that the common reasons for a database to be in “SUSPECT” mode always start from problems related to the physical structure of the files that make up the database in SQL Server. We also checked the importance of looking for the source of the corruption / failure of the database file in the server logs (both Windows and SQL Server). We reinforce how important it is to have a backup plan tested in the production environment in case any damage occurs, the data loss is as little (if any) as possible.

Here are some tips for deciding between restoring a database or trying to recover it.

We exemplify two types of datafile corruption, one being that we recover the physically corrupted data log (with the HxD application) and another example with the data log without recovery. In this last example, we see several options to return the database to the “ONLINE” state and leave it enabled for access, even if the log file is no longer present.

In all cases, it is really important to remind the organization that downtime of the environment and possible loss of data will be necessary, so it is important to stay calm, get organized and work on the information that SQL Server (and Windows) has to discover the source of the problem and how to solve it.

In the action plan for this problem, assess whether it depends on a quick fix (like rebuilding an index only) or really depends on a restoration of the database backup, always taking into account the time you will need to leave the databases (or database) inaccessible, remembering that this can affect the company's business.

Database corruption cases will always end up hindering the business flow and, in extreme cases, the company's revenue, as the loss of data will result in systems that fail to calculate or present information, in addition to user data end lost.

Always keep in mind that prevention is better than having to recover a base with corruption. Therefore, periodically run the DBCC CHECKDB command on the databases and interpret its results. If you have large databases in production and the DBCC CHECKDB command generates performance degradation, use the PHYSICAL_ONLY option. Microsoft recommends for environments where the volume of information is significant. This option makes the command faster and does not affect the server, however it does not check the FILESTREAM data, which can be checked later.

And if you find yourself in the worst case scenario where there is total data loss and where the problem database has no backup to restore, besides all the recovery attempts we used here have failed, there is only one way out: re-entry of all data manually into a new database.

Links

* <http://support2.microsoft.com/kb/967351/en-us> (A heavily fragmented file in an NTFS volume may not grow beyond a certain size)

* <http://mh-nexus.de/en/hxd/> (HxD software)