SQL Server Database Corruption: o que fazer para evitar

Este artigo apresenta piores práticas, causas comuns, propagação da corrupção, mecanismos de detecção e alerta de corrupção da base de dados. Além disso, serão apresentadas alternativas de solução para o problema.

Fique por dentro

A corrupção do banco de dados envolve a perda de informações levando seu banco a um estado inconsistente. Neste artigo serão abordadas as piores práticas, causas comuns, propagação da corrupção, mecanismos de detecção e alerta. Além disso, mostraremos como proceder para realizar a recuperação de página de dados, verificação de consistência, backups com verificação, sinais de corrupção, restauração e reparação, recuperação por backup e correção. O tema discutido neste artigo é útil porque irá lhe ajudar a encontrar soluções para situações extremamente críticas no dia a dia de um DBA.

Corrupções de dados acontecem a todo momento e seus motivos vão de problemas aleatórios a falhas no subsistema de I/O. Paul Randal, um dos maiores especialistas na área, sempre diz que as pessoas que nunca passaram por um cenário de corrupção durante a carreira são sortudas, pois em algum momento inevitavelmente irão passar por ela.

Sabendo que corrupções acontecem bastante, a questão principal é estar preparado para quando acontecer e ter a capacidade de identificar o quanto antes, pois em muitas situações ela só é encontrada tarde demais, quando já tomou proporções enormes e o estrago já foi feito. Esse tipo de caso acontece, por exemplo, pela falta de preparo em sua identificação ou o não entendimento dos alertas que o subsistema de I/O gera.

Levando em consideração estas informações, temos dois pontos principais:

· Quanto mais rápido a corrupção for identificada, mais facilmente você será capaz de efetuar a recuperação com menor inatividade e perda de dados;

· As pessoas não sabem o que fazer depois que detectam uma corrupção, o que leva

diretamente a uma chance maior de perda de dados e tempo de inatividade do que é necessário.

Esses dois pontos são muito importantes, porque também impactam muitas vezes na saúde financeira da corporação, podendo levar ao afastamento do DBA.

Piores práticas

Muitos DBAs não têm ideia do que fazer quando uma corrupção de dados acontece. Em cenários reais, essa situação vem em conjunto com a pressão imediata dos superiores. Dessa maneira, processos de recuperação podem acabar sendo executados incorretamente, utilizando as piores práticas.

Uma das práticas mais comuns após a corrupção ser identificada é a reinicialização do SQL Server, em uma tentativa de possível correção após o serviço iniciar novamente. Essa prática, na grande maioria das vezes, irá apenas atrasar o processo de recuperação devido ao tempo de reinicialização.

Outra situação bem comum é partir para o último recurso de recuperação e causar a perda de dados. Por exemplo, executar a recuperação com a opção que permite a perda de dados sem avaliar a utilização dos backups ou, ainda pior, saber que existe uma corrupção, mas não utilizar o DBCC CHECKDB para identificar o tipo e a melhor opção para correção.

Remover o log de transação ou a base corrompida também são casos que já foram vistos por DBAs achando que ajudariam a corrigir a corrupção, quando na verdade só pioram mais levando a novos problemas.

Causa

É importante entender as diferentes causas de uma corrupção e porque é quase inevitável que ela aconteça. Em 99% das vezes é o subsistema de I/O que causa a corrupção. Pode-se entender como subsistema de I/O qualquer componente que esteja abaixo do SQL Server, sendo assim, as principais causas são as seguintes:

· Sistema Operacional: quando o SQL Server inicialmente escreve uma página de dados, ele pede ao Windows para que faça esse trabalho;

· File system: a execução é feita em cima de um sistema de arquivos, geralmente NTFS. Assim, podem haver *File System Filter Drivers* de terceiros instalados como antivírus, criptografia, etc;

· Placas de Rede/Switches/Cabos: a página será transportada através da infraestrutura de rede;

· Controlador de SAN/RAID: para acessar o disco, a página ainda terá que passar pela controladora do SAN (*storage area network*) e controladora do RAID (*redundant array of independent disks*).

Todo esse caminho demonstra a intensa movimentação que existe no subsistema de I/O, com várias partes móveis e códigos envolvidos. Toda vez que existem códigos escritos por desenvolvedores, existe um potencial de bug e as coisas podem dar errado. É por isso que existem os Services Packs, atualizações cumulativas para o SQL Server e também as atualizações no seu firmware.

Além dos 99% dos casos no qual a corrupção é causada pelo subsistema de I/O, existe o outro 1% com as possíveis causas:

· Corrupção de memória: páginas de dados do SQL Server são armazenadas no buffer pool, que é basicamente um grande bloco de memória cache. Dessa forma, os chips de memória que realmente armazenam aquele dado dentro do servidor podem falhar.

Se um chip de memória falha, pode causar uma corrupção a uma porção de dados mantida na memória;

· Bugs do SQL Server: o SQL Server é um grande produto, tendo milhões de linhas de códigos. Ocasionalmente existem bugs no SQL Server que podem causar corrupção;

· Erro humano: essa é uma referência a uma pessoa fazendo algo que não deveria fazer manualmente. Por exemplo, colocar o banco de dados offline e editar o arquivo de dados do SQL Server ou excluir o arquivo de log de transações. Quando o serviço subir novamente, as transações ativas não poderão ser revertidas, causando corrupção de dados.

Existem muitos equívocos no entendimento dos causadores de corrupção, assim como, dos não causadores. Vejamos quais são eles:

· Qualquer coisa que a aplicação possa fazer;

· Qualquer operação que você possa fazer no SQL Server, desde que, com os comandos suportados e documentados;

· Interromper um shrink na base de dados, rebuild de índice ou queries de longa duração;

· Parar o serviço do SQL Server, a menos que exista algo ocorrendo por trás do SQL Server. Na realidade, o SQL Server é projetado para sofrer falhas e conseguir voltar, é por isso que temos o log de transações.

Propagação da corrupção

Nenhuma das tecnologias de redundância que o SQL Server tem (database mirroring, log shipping, replication e always on) vão propagar corrupção de arquivos de dados

causados pelo subsistema de I/O. Esses recursos têm em comum o fato de todos enviarem registros ou blocos de transação e não páginas dos arquivos de dados do disco, sendo assim, caso o subsistema de I/O corrompa uma página em disco, essas tecnologias não enviarão a corrupção a um subsistema de I/O remoto.

Corrupção misteriosa

Esse fenômeno não é muito comum, mas existem situações nas quais uma corrupção é detectada em uma rotina automatizada que faz verificações de consistência. Entretanto, ao efetuar uma segunda verificação, a corrupção não aparece, levando a crer que o comando DBCC CHECKDB está com falha ou simplesmente deixando uma incógnita. Imagine o seguinte cenário: existe uma série de rotinas automáticas sendo executadas durante a madrugada, sendo que, entre elas temos uma verificação de integridade e uma manutenção de índice. O DBA recebe uma notificação no dia seguinte informando que uma corrupção foi detectada. Logo em seguida, ele executa o comando DBCC CHECKDB para analisar o tipo de corrupção, mas como foi dito antes, a corrupção não aparece.

A corrupção não desapareceu simplesmente na primeira vez que foi relatado, realmente a mesma tinha acontecido e provavelmente com uma notificação de corrupção em páginas atribuídas a um índice. Em seguida, a verificação de integridade, alguma outra rotina de manutenção de índice foi executada, ou seja, o índice que estava corrompido foi reconstruído e quando um índice é reconstruído ele monta uma nova estrutura de índice com novas páginas no arquivo de dados e, em seguida, tira as alocações das antigas páginas dos índices. Assim, eles não fazem mais parte da tabela.

Dessa forma, quando uma segunda verificação de integridade é feita, nenhuma inconsistência será encontrada no índice original, pois o mesmo foi reconstruído em

uma rotina executada depois e novas páginas de dados estão sendo lidas.

Mecanismos de detecção e alerta

O SQL Server possui vários mecanismos integrados que lhe permite detectar e alertar automaticamente quando problemas de corrupção são encontrados durante operações de I/O. Existem quatro pontos que serão abordados nos tópicos em seguida:

· Proteção de Página - Torn-page Detection;

· Proteção de Página - Page Checksum;

· Diferentes tipos de erros de I/O;

· Monitoramento de erros de I/O.

Opções de proteção de página

O SQL Server possui uma maneira de manter protegidas as páginas de dados que estão no disco como um meio rápido de detectar corrupção quando uma página for lida na memória. As configurações de proteção a página de dados podem ser habilitadas ou alteradas, assim como mostra a **Listagem 1**.

Listagem 1. Alteração do nível de proteção da página de dados.

```
USE MASTER
GO
/*COLOQUE O NOME DO BANCO DE DADOS E O TIPO DE PROTEÇÃO OPTADO*/
ALTER DATABASE “NOME BD” SET PAGE_VERIFY “TIPO DE PROTEÇÃO”
```

Existem três tipos de opção para utilizar como configuração: checksum, torn-page detection ou nenhuma. Não importa qual tipo de opção você escolher, todo o trabalho é

feito pela área do buffer pool, que é o bloco de memória onde o SQL Server mantém cópias das páginas do arquivo de dados.

Torn-Page Detection

No SQL Server, toda página de dados tem o tamanho de 8 Kb, sendo dividida em 16 blocos de discos com 512 bytes. Para uma página ser corretamente escrita no disco, cada um desses 16 blocos deve ser escrito de forma adequada, mas claro que é possível que uma página não seja escrita corretamente como, por exemplo, se a energia falhar ou o disco não conseguir capacidade suficiente para terminar de escrever esses 16 blocos. Torn-Page é exatamente quando acontece algum desses casos citados.

O SQL Server tem um mecanismo que permite detectar quando um Torn-Page acontece, sendo feito através de cálculos com os bits dos blocos e os cabeçalhos das páginas. Com esse mecanismo o SQL Server é capaz de detectar se uma página não foi completamente escrita.

Esse método é muito bom para detectar quando um Torn-Page ocorre, mas não permite ao SQL Server identificar quando aconteceu um problema ou uma corrupção dentro de um setor de disco.

Page Checksum

O Torn-Page Detection não pode detectar problemas que ocorrem no meio dos blocos de disco e é exatamente isso que o Page Checksum faz. Essa opção de verificação por página foi introduzida no SQL Server 2005 e todas as bases criadas a partir dessa versão já estão com essa configuração habilitada por padrão, menos o TempDB que teve que esperar até o SQL Server 2008 para poder também trabalhar com o Page

Checksum.

Assim como o Torn-Page Detection, o Page Checksum também atua no buffer pool e sua lógica é simples. Há um valor de 4 Bytes que é calculado para o Page Checksum e guardado no cabeçalho da página, sendo este o último passo que o SQL Server realiza quando vai escrever uma página no disco. Quando uma página for lida novamente a partir do disco para memória, a primeira coisa que será feita é o recálculo do Page Checksum para verificar se o valor é igual ao que está armazenado na página. Caso o Page Checksum não esteja correto, o SQL Server sabe que algo abaixo dele corrompeu a página. Uma boa prática é sempre utilizar essa opção para os bancos de dados.

Uma página é verificada pelo Page Checksum nas seguintes condições:

· Sempre que é lida pelo Buffer Pool, seja normalmente ou por alguma rotina de verificação de consistência;

· No procedimento de backup, mas nesse caso o Buffer Pool não é utilizado, um canal próprio para os dados/log é aberto e sua leitura é feita diretamente;

· Se um backup tiver sido criado com a opção do Page Checksum habilitada, sempre que as páginas forem lidas a partir do backup para uma operação de restauração ou para verificar o conteúdo da cópia de segurança, o Page Checksum será recalculado.

Na **Listagem 2** são demonstradas as opções de proteção de página. Note que efetuamos uma consulta à tabela de sistema para verificar a configuração de proteção de página. O valor padrão é herdado do banco de sistema model. Em seguida, para alterarmos o tipo de configuração de proteção de páginas, devemos executar a instrução a nível de banco de dados.

Listagem 2. Demonstração das opções de proteção de página

```
USE MASTER
GO
/*CRIA UMA NOVA BASE DE DADOS.
OBS: CRIE ANTES O DIRETÓRIO PARA OS ARQUIVOS OU REDIRECIONE PARA OUTRO LOCAL.*/
CREATE DATABASE EMPRESA ON PRIMARY (
 NAME = 'EMPRESA',
 FILENAME = 'C:\DEMO_CORRUPCAO\EMPRESA.MDF')
LOG ON (
 NAME = 'EMPRESA_LOG',
 FILENAME = 'C:\DEMO_CORRUPCAO\EMPRESA_LOG.LDF')
GO
SELECT
 NAME,
 PAGE_VERIFY_OPTION,
 PAGE_VERIFY_OPTION_DESC
FROM
 SYS.DATABASES
WHERE
 NAME IN ('EMPRESA','MODEL')
GO
ALTER DATABASE EMPRESA
SET PAGE_VERIFY TORN_PAGE_DETECTION
GO
ALTER DATABASE EMPRESA
SET PAGE_VERIFY CHECKSUM
```

Adicionalmente também pode ser alterada a configuração da proteção de página via interface do SSMS, acessando as propriedades do banco de dados, assim como mostra o exemplo da **Figura 1**.

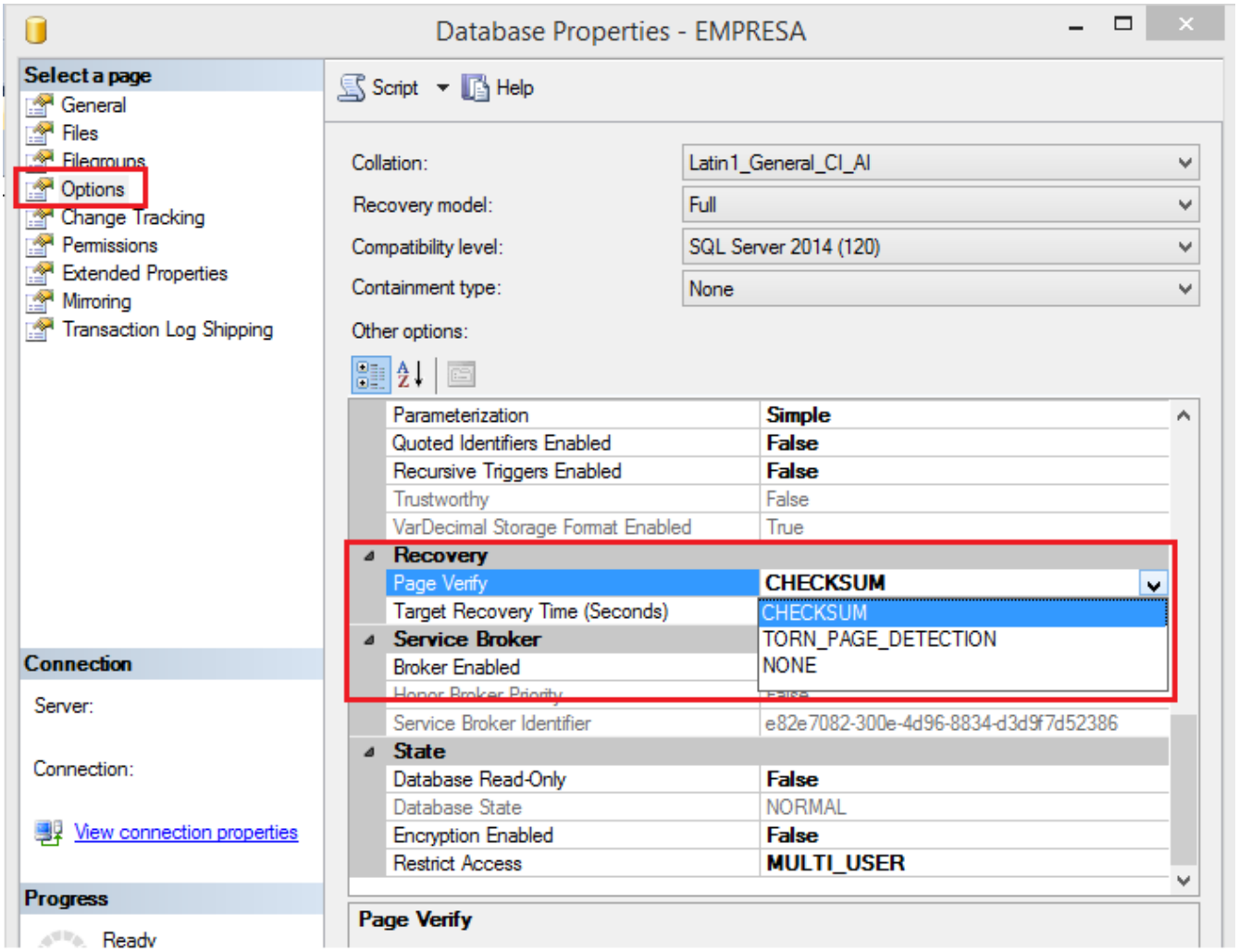


Figura 1. Alteração da proteção de página via SSMS

Erros de I/O

Existem três tipos de erros de I/O que o SQL Server irá retornar quando é detectado um problema:

· Código 823: representa a existência de um erro no subsistema de I/O. O SQL Server tenta uma operação solicitando ao Windows para acessar algumas páginas e é negado, não permitindo ao subsistema de I/O o acesso requerido. A severidade desse erro é 24, sendo a maior que é possível de se obter e indica exatamente um problema fatal. Como resultado, algumas ações são executadas pelo SQL Server:

o A conexão será quebrada;

o A tabela de sistema msdb.dbo.suspect_pages será carregada. Ela tem como objetivo guardar informações das páginas que foram consideradas suspeitas de corrupção;

o Um registro será adicionado no log de erros do SQL Server e no log de eventos da aplicação no Windows.

· Código 824: nesse caso, a falha na consistência lógica ocorre após a operação de I/O, ou seja, o Windows entregou os dados requeridos e o SQL Server detectou problemas. A severidade desse erro também é 24;

· Código 825: conhecido como erro de Read-Retry, não é tão recorrente quanto os dois primeiros, mas é visto em algumas circunstâncias. Esse erro significa que qualquer falha no I/O que é detectado pelo buffer pool irá fazer com que essa tentativa de leitura seja repetida automaticamente até 4 vezes. Caso na quinta tentativa o erro de I/O ainda persista, será visto o erro 823 ou 824, mas se umas das tentativas anteriores obtiver sucesso, você não verá um erro de I/O, apenas 825. O problema desse erro é que seu nível de severidade é 10, ou seja, retorna apenas uma mensagem informativa, podendo passar facilmente despercebido em uma verificação. Assim, existe a recomendação que se crie um alerta específico para ele;

O Read-Retry é considerado um sinal de alerta para problemas muito mais graves que podem vir na sequência, porque significa que o subsistema de I/O está retornando dados incorretos para o SQL Server e dessa forma ele é forçado a fazer o Read-Retry.

As **Listagens 3** a **5** mostram como se parecem as estruturas das mensagens de erro 823, 824 e 825.

Listagem 3. Exemplo da mensagem de erro de código 823

```
Msg 823, Level 24, State 2, Line 81
```

```
The operating system returned error 1006(The volume for a file has been externally
This is a severe system-level error condition that threatens database integrity ar
Complete a full database consistency check (DBCC CHECKDB). This error can be cause
```

Listagem 4. Exemplo da mensagem de erro de código 824

```
Msg 824, Level 24, State 2, Line 2
SQL Server detected a logical consistency-based I/O error: incorrect checksum (exp
 Additional messages in the SQL Server error log or system event log may provide m
```

Listagem 5. Exemplo da mensagem de erro de código 825

```
Msg 825, Level 10, State 2, Line 1.
A read of the file 'PRODUCAO.MDF' at offset 0x00000000012000 succeeded after failir
```

Em todas as mensagens, existem informações específicas que dão uma certa orientação do que ocorreu e o que deve ser feito, como o ID do banco afetado, o arquivo que apresentou problema, o nível de severidade do erro, entre outros.

Para poder simular as corrupções nos exemplos que serão feitos ao longo desse artigo, usaremos o comando não documentado DBCC IND e DBCC WritePage, sendo que este último é considerado o comando mais perigoso do SQL Server devido ao seu impacto e existem uma série de restrições em seu uso como, por exemplo, nunca utilizar em um ambiente de produção. O DBCC WritePage permitirá simular uma corrupção, pois poderemos danificar uma página dados e alterar estruturas dentro do SQL Server.

A **Listagem 6** mostra um exemplo de como um erro é apresentado na prática.

Inicialmente criamos uma nova tabela e inserimos alguns valores. Em seguida, listamos todas as páginas de dados da tabela com o comando DBCC IND e escolhemos uma para causar a corrupção. Para isso, memorize o PAGEPID da página selecionada. O comando DBCC WRITEPAGE irá substituir os dois primeiros bytes na página de dados no disco. Utilize o PAGEPID selecionado anteriormente na terceira posição dos parâmetros. Em seguida, limpamos o log de erros e páginas suspeitas na instância para observarmos melhor os erros que serão gerados. Feito isso, agora executamos a consulta para retornar o erro de I/O 824. Repare que ele é igual ao mostrado no exemplo. Por fim, consultamos a tabela de páginas suspeitas, onde é possível ver uma entrada de registro para cada página corrompida que foi encontrada e a quantidade de vezes que isso ocorreu.

Listagem 6. Simulação da corrupção

```
USE EMPRESA
GO
CREATE TABLE DADOS(
 ID INT IDENTITY,
 VALOR CHAR (8000) DEFAULT 'A'
)
GO
INSERT INTO DADOS DEFAULT VALUES
GO 10
DBCC IND ('EMPRESA', 'DADOS', -1)
GO
ALTER DATABASE EMPRESA SET SINGLE_USER
GO
DBCC WRITEPAGE ('EMPRESA', 1, 144, 0, 2, 0x0000, 1)
GO
ALTER DATABASE EMPRESA SET MULTI_USER
GO
DELETE FROM MSDB.DBO.SUSPECT_PAGES
EXEC SP_CYCLE_ERRORLOG
GO
SELECT
*
FROM EMPRESA.DBO.DADOS
GO
```

```
SELECT
*
FROM MSDB.DBO.SUSPECT_PAGES
/*OS VALORES DA COLUNA EVENT_TYPE ESTÃO DOCUMENTADOS NO BOOKS ONLINE DA MICROSOFT*
```

A **Figura 2** mostra o exemplo dos valores que são gravados na tabela de páginas suspeitas no MSDB.

Results | Messages

	database_id	file_id	page_id	event_type	error_count	last_update_date
1	22	1	144	2	1	2015-06-02 22:13:58.493

Figura 2. Consulta a tabela de páginas suspeitas

Adicionalmente, podemos ver no log de erros do SQL Server e nos eventos do Windows o erro gerado (ver **Figuras 3** e **4)**.

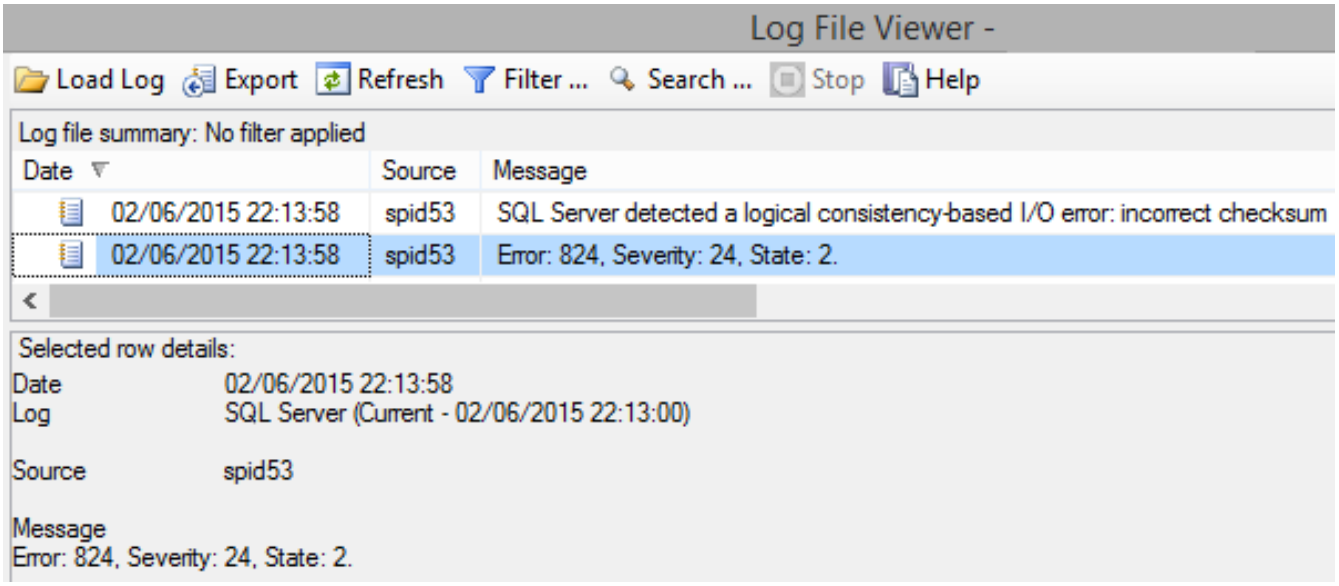
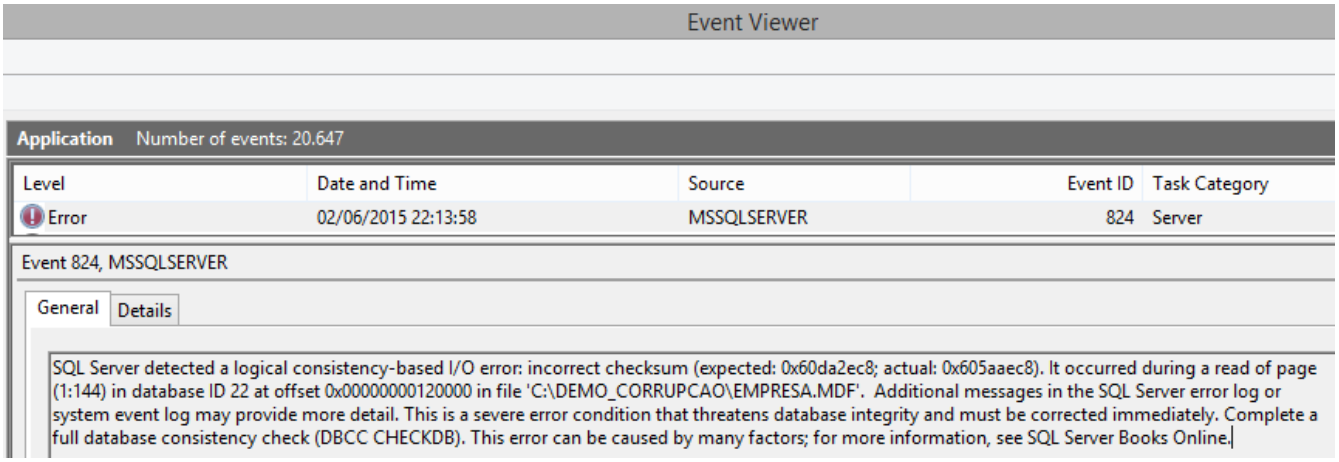


Figura 3. Log de erros do SQL Server



abrir imagem em nova janela

Figura 4. Log de eventos do Windows

Monitoramento de erro de I/O

Não há nenhum sentido na criação de Page Checksums se você não está percebendo quando os problemas estão ocorrendo. O log de erros do SQL Server pode ser verificado, mas esse tipo de operação manual é muito demorado e é propenso a ser esquecido. O melhor a ser feito é utilizar um processo automatizado de monitoramento. Dessa forma, se algum problema ocorrer, você será notificado imediatamente.

O recurso de alerta do SQL Server Agent pode ser utilizado com o intuito de envio de notificações como, por exemplo, se o erro de código 823 ocorrer, um e-mail pode ser enviado para o DBA. Outros mecanismos como Microsoft SCOM e ferramentas de terceiros também podem ser utilizados como solução de monitoramento.

Seja qual for o mecanismo utilizado para monitoramento, crie alertas para cada nível de severidade acima do 19, além de colocar para o erro 825.

Recuperação automática da página

As duas funcionalidades Database Mirroring e Availability Groups possuem um recurso

chamado Reparo Automático de Página que permite que algumas corrupções sejam reparadas automaticamente, sempre que possível. Isso funciona para alguns dos erros 823, 824 e às vezes 829.

As páginas corrompidas no servidor principal ou no espelho, assim como, na réplica primária e secundária, podem ser reparadas e esse mecanismo é possível, pois qualquer instância que tenha sido atingida pela página corrompida pode pedir ao parceiro a sua cópia da página. Caso seja bem-sucedido, a página é reparada com êxito.

Existem alguns tipos de páginas que não podem ser reparadas como File Header Pages, Boot Pages e alguns dos Allocation Bitmap do banco de dados inteiro. Mas a maioria das páginas podem ser automaticamente reparadas.

Os reparos acontecem de forma assíncrona, sendo assim, não será possível acessar uma página corrompida até que seja recuperada. Toda vez que uma recuperação automática é acionada, a página que está sendo recuperada é marcada com 829 para que consultas posteriores na página não disparem novos processos de recuperação.

Verificação de consistência

O SQL Server pode detectar automaticamente a corrupção através das opções de proteção de página, mas isso só funciona se as páginas corrompidas forem lidas do disco. Pode acontecer de a corrupção ser em partes do banco que são raramente acessadas e você não irá saber disso. Portanto, há um método que pode ser utilizado para forçar todas as páginas alocadas no banco serem lidas, a verificação de consistência.

É por esse motivo que é necessário que as verificações de consistências sejam executadas frequentemente, para ajudar o SQL Server a detectar as corrupções o mais

rapidamente possível e dessa forma, se recuperar com o mínimo de tempo de inatividade e perda de dados.

A verificação de consistência no contexto do SQL Server significa algum processo que valida a integridade do banco de dados, ou seja, o comando DBCC CHECKDB e suas variações.

Algumas dúvidas são muito comuns quando é abordado o uso do comando DBCC CHECKDB. Entraremos nas principais questões debatidas.

Quais bancos de dados devem ser verificados?

A verificação de consistência deve ser executada em todos os bancos de dados, inclusive nos de sistema, pois os impactos que podem ser causados por uma corrupção são grandes, podendo encerrar e interromper a inicialização da instância em algumas situações de falha na Master ou TempDB.

Como executar?

Existem basicamente três maneiras de executar a verificação de consistência: script automatizado pelo SQL Agent, manualmente e pelo plano de manutenção do SQL Server.

Com qual frequência executar?

A resposta para essa pergunta é relativa e depende de vários fatores como:

o Com o subsistema de I/O apresentando problemas constantes, você provavelmente vai querer executar as verificações de consistência com mais frequência do que se ele estivesse estável;

o Com uma estratégia de backup muito abrangente e bem testada, você pode não precisar executar a verificação de consistência tão recorrente;

o O tempo de inatividade e quantidade de perda de dados que é admissível caso ocorra uma corrupção deve ser levado em consideração na frequência de execução da verificação de consistência, pois irá determinar se será possível recuperar o ambiente dentro da margem de tempo e perda estabelecidos;

o Executar a verificação de consistência gasta muitos recursos de I/O e CPU, sendo assim, a janela de execução dessa rotina deve ser em um período no qual o sistema não seja comprometido em sua carga de trabalho;

o Para ambientes distintos, a frequência da verificação de consistência deve ser diferente. Na produção você provavelmente vai querer executar a rotina com mais frequência do que em um ambiente de teste ou desenvolvimento.

Lembre-se destes pontos no momento de avaliar a frequência da rotina de verificação de consistência, mas de maneira geral, é recomendo que seja executada pelo menos uma vez por semana.

É possível ter garantia?

Não importa quantas vezes você execute o processo de verificação de consistência, não é possível obter uma garantia de que seu banco está livre de corrupção em todos os momentos.

O comando DBCC CHECKDB, em sua execução, apenas faz a leitura de algumas páginas de dados por vez, pois não é possível que a base seja lida inteira ao mesmo tempo. Sendo assim, é feita a leitura página a página e seu processamento até o fim da execução. Dessa forma, existe a possibilidade que ocorra uma corrupção em

páginas já lidas previamente dentro do processo antes mesmo que ele seja finalizado. Essa corrupção não será detectada nesse processo de verificação de consistência que já está em andamento, sendo necessária uma nova execução para ter uma garantia de que tudo está certo.

Como verificar a consistência de um banco de dados grande?

O processo de verificação de consistência em um grande banco de dados com vários terabytes pode levar muitas horas para concluir e existem basicamente três maneiras para reduzir o tempo de execução do processo:

o Podem ser usadas as opções adicionais do DBCC CHECKDB para reduzir o tempo e recursos necessários para execução da verificação de consistência;

o É possível quebrar a verificação ao longo do tempo utilizando algumas das opções adicionais DBCC CHECKDB;

o Por último, pode ser utilizada uma instância do SQL Server em outro servidor para executar as verificações de consistência no banco de dados. Dessa forma, a carga de trabalho pode ser removida completamente do ambiente principal.

Para utilizar o procedimento de verificação de consistência usando outro servidor, faça backup full da base de dados no servidor que a verificação de consistência deveria ser executada e lembre-se de utilizar a opção de Checksum no procedimento. Em seguida, restaure a base de dados em um segundo servidor também utilizando a opção de Checksum e execute a verificação de consistência na base de dados restaurada.

Caso uma corrupção seja encontrada, ficaremos na dúvida se o problema foi no subsistema de I/O nesse segundo servidor, se é o arquivo de backup que tem a

corrupção ou o banco de dados no servidor principal. Uma vez que a corrupção é encontrada utilizando esse método de restauração em um servidor secundário, você será forçado a executar a verificação de consistência no servidor principal para identificar exatamente qual o problema.

As réplicas secundárias do Database Mirror ou Availability Group também costumam ser utilizadas como maneira de eliminar a carga de trabalho no servidor principal, executando a verificação de consistência nelas, mas isso não é valido. Os servidores das réplicas secundárias utilizam diferentes subsistemas de I/O e a corrupção não é propagada para servidores remotos. A verificação de consistência deve idealmente ser executada em todas as cópias do banco de dados.

Backup Checksum

Para utilizar a opção de checksum durante uma operação de backup, basta colocar o comando *with checksum* como opção adicional na criação da cópia de segurança.

Ao utilizar essa opção, todas as páginas que estão sendo lidas do banco de dados serão validadas e caso algum problema seja encontrado, a operação de backup será interrompida. Além disso, também é calculado o checksum ao longo de toda a operação e gravado no cabeçalho do backup.

Esse processo, além de praticamente não causar impactos significantes na CPU, é uma garantia extra de que não existem problemas de I/O na base que está sendo feito o backup, sendo assim, é recomendado sempre utilizar backup checksum.

Quando é verificada a integridade do arquivo de backup através do comando Restore VerifyOnly, também é possível utilizar adicionalmente o comando Checksum, caso esteja presente na mídia de backup. Esse tipo de validação é muito importante, pois os backups são suscetíveis à corrupção gerada pelo subsistema de I/O, assim como

qualquer outro arquivo de dados.

Uma vez que a opção checksum foi utilizada na operação de backup, durante a restauração, as páginas com checksum também podem ser verificadas desde que seja adicionado o comando na instrução. A restauração irá falhar caso exista uma inconsistência. As **Listagens 7** e **8** demonstram o uso do backup checksum.

Na primeira listagem, inicialmente criamos novamente o banco de dados empresa e inserimos alguns valores na tabela. Em seguida, escolhemos o PAGEPID de algumas páginas de dados após executarmos o DBCC IND. A próxima etapa será justamente corromper a página utilizando o comando DBCC WRITEPAGE. Feito isso, criaremos diferentes tipos de backup. Contudo, sabemos que no banco existe uma corrupção, mas ao fazermos um backup regular, sem nenhum tipo de verificação, as páginas corrompidas são colocadas no arquivo de backup sem problemas. Por outro lado, ao utilizarmos a opção WITK CHECKSUM, o SQL Server irá retornar uma mensagem informando que foi encontrada uma corrupção no arquivo de dados, que é o comportamento que esperamos que aconteça quando o backup apresente problemas.

Listagem 7. Demonstração de uso do backup Checksum – Parte 1

```
USE MASTER

GO
IF DB_ID('EMPRESA') IS NOT NULL
ALTER DATABASE EMPRESA SET SINGLE_USER WITH ROLLBACK IMMEDIATE
GO
DROP DATABASE EMPRESA
GO
CREATE DATABASE EMPRESA ON PRIMARY (
 NAME = 'EMPRESA',
 FILENAME = 'C:\DEMO_CORRUPCAO\EMPRESA.MDF')
LOG ON (
 NAME = 'EMPRESA_LOG',
 FILENAME = 'C:\DEMO_CORRUPCAO\EMPRESA_LOG.LDF')
GO
```

```
USE EMPRESA
GO
CREATE TABLE DADOS(
ID INT IDENTITY,
VALOR CHAR (8000) DEFAULT 'A')
GO
INSERT INTO DADOS DEFAULT VALUES
GO 10
DBCC IND ('EMPRESA', 'DADOS', -1)
GO
ALTER DATABASE EMPRESA SET SINGLE_USER
GO
DBCC WRITEPAGE ('EMPRESA', 1, 121, 0, 2, 0X0000, 1)
GO
ALTER DATABASE EMPRESA SET MULTI_USER
GO
BACKUP DATABASE EMPRESA
TO DISK = 'C:\DEMO_CORRUPCAO\EMPRESA.BAK'
WITH INIT
GO
BACKUP DATABASE EMPRESA
TO DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_CHECKSUM.BAK'
WITH INIT, CHECKSUM
```

Na **Listagem 8** vemos inicialmente que podemos forçar a execução do backup mesmo que ele esteja danificado e a com a opção CHECKSUM. Para isso, deve ser utilizada a opção CONTINUE_AFTER_ERRO. O motivo pelo qual não removemos a opção CHECKSUM para concluirmos o backup que está danificado é porque queremos saber essa informação. Em seguida, para verificar a integridade do arquivo de backup, podemos utilizar a opção RESTORE VERIFYONLY.

Ainda nesta listagem, vemos também que no processo de restauração a partir do arquivo de backup regular, não serão apresentados problemas apesar de sabermos que existem corrupções nesse arquivo. Porém, ao restauramos o arquivo de backup feito com o CHECKSUM com essa mesma opção de restore, o processo será interrompido por uma mensagem de erro informando sobre a corrupção existente no arquivo. Removendo a opção de CHECKSUM no restore, a mensagem de erro

permanece, ou seja, o erro ainda é detectado na restauração. Para forçar um backup corrompido ser restaurado, deve ser utilizada a opção CONTINUE_AFTER_ERRO em conjunto com a instrução restore. Observem que uma mensagem de alerta será apresentada na conclusão informando sobre os danos encontrados no arquivo.

Listagem 8. Demonstração de uso do backup Checksum – Parte 2

```
GO
BACKUP DATABASE EMPRESA
TO DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_CHECKSUM.BAK'
WITH INIT, CHECKSUM, CONTINUE_AFTER_ERROR

GO
/*VEJA O CABEÇALHO DO ARQUIVO DE BACKUP CRIADO.
O CAMPO HasBackupChecksums E IsDamaged RETORNA ESSAS INFORMAÇÕES PARA O ARQUIVO CC
RESTORE HEADERONLY
FROM DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_CHECKSUM.BAK'

GO
/*SEM A OPÇÃO DE CHECKSUM O BACKUP SERÁ CONSIDERADO VÁLIDO.*/
RESTORE VERIFYONLY
FROM DISK = 'C:\DEMO_CORRUPCAO\EMPRESA.BAK'
GO
/*PARA O BACKUP QUE FOI FORÇADO A EXECUÇÃO, O RETORNO É UMA MENSAGEM INFORMANDO QU
RESTORE VERIFYONLY
FROM DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_CHECKSUM.BAK'
GO
/*USANDO A OPÇÃO WITH CHECKSUM PARA COMPLEMENTAR A VERIFICAÇÃO NÃO SERÁ POSSÍVEL M
RESTORE VERIFYONLY
FROM DISK = 'C:\DEMO_CORRUPCAO\EMPRESA.BAK'
WITH CHECKSUM
GO
/*PARA O ARQUIVO QUE FOI GERADO UTILIZANDO O CHECKSUM, A VERIFICAÇÃO IRÁ RETORNAR
RESTORE VERIFYONLY
FROM DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_CHECKSUM.BAK'
WITH CHECKSUM

GO
RESTORE DATABASE EMPRESA_COPIA
FROM DISK = 'C:\DEMO_CORRUPCAO\EMPRESA.BAK'
WITH
 MOVE 'EMPRESA' TO 'C:\DEMO_CORRUPCAO\EMPRESA_COPIA.MDF',
```

```
 MOVE 'EMPRESA_LOG' TO 'C:\DEMO_CORRUPCAO\EMPRESA_COPIA_LOG.LDF',
 REPLACE

GO
RESTORE DATABASE EMPRESA_COPIA
FROM DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_CHECKSUM.BAK'
WITH
 MOVE 'EMPRESA' TO 'C:\DEMO_CORRUPCAO\EMPRESA_COPIA.MDF',
 MOVE 'EMPRESA_LOG' TO 'C:\DEMO_CORRUPCAO\EMPRESA_COPIA_LOG.LDF',
 REPLACE, CHECKSUM

GO
RESTORE DATABASE EMPRESA_COPIA
FROM DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_CHECKSUM.BAK'
WITH
 MOVE 'EMPRESA' TO 'C:\DEMO_CORRUPCAO\EMPRESA_COPIA.MDF',
 MOVE 'EMPRESA_LOG' TO 'C:\DEMO_CORRUPCAO\EMPRESA_COPIA_LOG.LDF',
 REPLACE

GO
RESTORE DATABASE EMPRESA_COPIA
FROM DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_CHECKSUM.BAK'
WITH
 MOVE 'EMPRESA' TO 'C:\DEMO_CORRUPCAO\EMPRESA_COPIA.MDF',
 MOVE 'EMPRESA_LOG' TO 'C:\DEMO_CORRUPCAO\EMPRESA_COPIA_LOG.LDF',
 REPLACE, CONTINUE_AFTER_ERROR
```

DBCC CHECKDB

O DBCC CHECKDB é o principal comando utilizado nas verificações de consistência. Na maioria das vezes, é ele que será usado ou então seus comandos relacionados que representam um subconjunto de funcionalidades adicionais para verificação.

O comando DBCC CHECKDB irá executar a verificação de consistência em um único banco de dados, mas tem uma exceção. Quando o DBCC CHECKDB é executado no banco master, ele também irá verificar o banco resource.

A única maneira de forçar todas as páginas alocadas no banco serem lidas é

executando o DBCC CHECKDB, apesar da operação de backup também conseguir isso. Mas se você está seguindo as melhores práticas e utilizando a opção de Checksum, o backup vai parar quando aparecer uma corrupção. A ideia por trás do DBCC CHECKDB é exatamente a capacidade de ler tudo no banco de dados, desde que não exista alguma corrupção que o interrompa. Dessa forma, ele força com que todas as pages checksums que você tenha nas páginas do banco de dados sejam lidas.

Um detalhe que será notado na execução do DBCC CHECKDB é o seu gasto de recursos. Ele é muito intenso, ou seja, usa muito processamento, memória e ainda mais I/O. São esses pontos de desempenho que levam as pessoas a terem problemas na execução da verificação de consistência em seus ambientes de produção.

Existem vários comandos adicionais que podem ser utilizados em conjunto com o DBCC, mudando o comportamento no processamento, tempo de execução, detalhes do que é verificado, entre outras coisas. Vejamos alguns desses comandos adicionais:

o DBCC CHECKALLOC: verifica a consistência das estruturas alocadas no banco de dados tais como as páginas GAM, SGAM, FPS e IAM, que são estruturas que mantêm o controle de quais páginas, extents, índices e tabelas estão alocados. A primeira etapa que esse comando irá realizar é a verificação de algumas tabelas de sistemas críticas, garantindo que essas tabelas possam ser lidas corretamente. É através delas que o DBCC CHECKALLOC consegue as informações sobre quais tabelas e índices devem ter suas estruturas de alocação verificadas. O comando adicional DBCC CHECKALLOC, assim como, o CHECKTABLE, CHECKCATALOG e DBCC CHECKFILEGROUP que serão detalhados a seguir, estão contemplados como parte do processo que o DBCC CHECKDB já executa;

o DBCC CHECKTABLE: verifica a consistência de uma única tabela e todos os seus

índices. Um dos processos executados é a leitura de cada página do arquivo de dados alocado na tabela, certificando-se de que aquela página possa ser lida sem nenhum erro de I/O e verificando se a sua estrutura é válida;

o DBCC CHECKCATALOG: no uso desse comando são verificadas as relações entre os catálogos do sistema, ou seja, as tabelas ocultas que armazenam todos os metadados sobre as tabelas, índices, colunas e outros.

o DBCC CHECKFILEGROUP: esse comando realiza verificações de consistência em um único Filegroup. Primeiramente são executadas verificações nas alocações para todos os bancos de dados e, em seguida, ele faz todas as análises de consistência lógica para apenas a tabela e partições do índice que estão no Filegroup que foi solicitado;

o DBCC CHECKIDENT: esse comando é usado para verificar o valor do Identity de uma tabela e se necessário, o propaga novamente ou restaura com um novo valor, sendo que esse comando não faz parte do DBCC CHECKDB;

o DBCC CHECKCONSTRAINTS: com esse comando, são verificadas as check constraints e chaves estrangeiras para certificar que elas são válidas, mas também não faz parte do DBCC CHECKDB. O DBCC CHECKCONSTRAINTS é um dos comandos que às vezes você precisa executar depois que uma operação de reparo é feita.

Argumentos do DBCC CHECK

Além das opções adicionais ao DBCC, o próprio comando DBCC CHECKDB pode ser utilizado em conjunto com vários argumentos que ampliam sua capacidade possibilitando outras formas de execução:

· NO_INFOMSGS: faz com que as mensagens informativas não sejam apresentadas

no retorno;

· ALL_ERRORMSGS: permite que todas as mensagens de erros sejam apresentadas. Essa opção existe apenas para versões anteriores ao SQL Server 2008, pois caso ela não fosse usada, apenas as 200 primeiras linhas de erro eram exibidas por padrão;

· NO_INDEX: apenas diz para não verificar qualquer índice NonClustered;

· DATA_PURITY: realiza a validação dos dados da coluna, certificando-se de que esses valores estão de acordo com os limites existentes para cada tipo de dados;

· ESTIMATEONLY: solicita ao CHECKDB para estimar quanto de espaço no TempDB será necessário para executar com sucesso o CHECKDB. Será fornecida uma estimativa bem conservadora do quão grande o seu TempDB tem que ser;

· TABLOCK: informar ao DBCC CHECKDB para usar bloqueios em nível de tabela no banco de dados em vez de criar um snapshot do mesmo para poder executar a verificação de consistência. Serão gerados bloqueios exclusivos no banco de dados para as verificações de alocação e, em seguida, bloqueios compartilhados para as verificações de tabela. A opção TABLOCK não pode ser usada no banco de dados master porque não é possível colocar um bloqueio exclusivo nesse banco;

· EXTENDED_LOGICAL_CHECKS: permite que validações extras possam ser feitas em views indexadas, índices XML e espaciais;

· PHYSICAL_ONLY: informa ao CHECKDB para ignorar a maioria das verificações de consistência lógica. Dessa forma, o processo é resumido a consultas às tabelas críticas de sistema, verificações de alocação e validações a nível estrutural das páginas atribuídas no banco de dados. Sendo assim, o tempo e recursos gastos na execução do DBCC CHECKDB são reduzidos.

No exemplo da **Listagem 9** mostramos os argumentos adicionais do DBCC CHECKDB mais comumente utilizados. Nesta listagem, inicialmente criamos o banco de dados empresa e inserimos alguns valores. Em seguida, executamos o comando DBCC CHECKDB sem opções adicionais para, na sequência, considerarmos a opção PHYSICAL_ONLY.

Listagem 9. Argumentos adicionais do DBCC CHECKDB

```
USE MASTER
GO
IF DB_ID('EMPRESA') IS NOT NULL
ALTER DATABASE EMPRESA SET SINGLE_USER WITH ROLLBACK IMMEDIATE
GO
DROP DATABASE EMPRESA
GO
/*CRIE O BANCO DE DADOS EMPRESA E INSIRA ALGUNS VALORES NA TABELA*/
CREATE DATABASE EMPRESA ON PRIMARY (
 NAME = 'EMPRESA',
 FILENAME = 'C:\DEMO_CORRUPCAO\EMPRESA.MDF')
LOG ON (
 NAME = 'EMPRESA_LOG',
 FILENAME = 'C:\DEMO_CORRUPCAO\EMPRESA_LOG.LDF')
GO
USE EMPRESA
GO
CREATE TABLE DADOS(
ID INT IDENTITY,
VALOR CHAR (8000) DEFAULT 'A')
GO
INSERT INTO DADOS DEFAULT VALUES
GO 10
SET STATISTICS TIME ON
GO
/*EXECUTE O DBCC CHECKDB SEM OPÇÕES ADICIONAIS*/
DBCC CHECKDB ('EMPRESA') WITH NO_INFOMSGS
GO
SET STATISTICS TIME OFF
/*GUARDE O TEMPO DE CPU E DECORRIDO
CPU TIME: ?
ELAPSED TIME: ? */
GO
/*AGORA EXECUTE TODOS OS COMANDOS NOVAMENTE, MAS COM A OPÇÃO PHYSICAL_ONLY INCLUSA
```

```
DBCC DROPCLEANBUFFERS
GO
SET STATISTICS TIME ON
GO
DBCC CHECKDB ('EMPRESA') WITH NO_INFOMSGS, PHYSICAL_ONLY
GO
SET STATISTICS TIME OFF
GO
/*GUARDE O TEMPO DE CPU E DECORRIDO
CPU TIME: ?
ELAPSED TIME: ?
VEJA QUE O TEMPO DECORRIDO (ELAPSED TIME) CONTINUA PRATICAMENTE O MESMO, MAS O TEM
```

Tempo de execução do DBCC CHECKDB

O tempo que leva a execução do DBCC CHECKDB é uma pergunta que é feita frequentemente e a resposta depende de alguns fatores, veja os principais:

· Tamanho da base de dados: o quão grande é o banco de dados e não nos referimos apenas ao tamanho do arquivo de dados, mas também a quantas páginas alocadas existem no banco, pois quanto maior a quantidade de páginas, mais tempo será necessário para concluir o processo;

· Carga de trabalho no servidor: em termos de I/O e CPU, quanto mais carga concorrente existe no servidor, mais tempo irá levar a execução, porque o DBCC CHECKDB gasta uma grande quantidade de recursos do servidor;

· Mudanças no banco de dados: o DBCC CHECKDB usa um snapshot do banco de dados em seu processo, sendo assim, qualquer mudança que é feita no banco é colocada no snapshot. Quanto mais mudanças estão acontecendo enquanto a verificação de consistência está em execução, mais tempo ela vai levar para concluir;

· Recursos de hardware do servidor: em termos de velocidade de I/O, quanto melhor as

capacidades do subsistema, mais rápida será a leitura de todas as páginas no banco de dados, assim como a verificação. Com relação ao número de CPUs, quanto maior a quantidade, provavelmente mais rápido será executada a verificação;

· Velocidades dos discos do TempDB: o TempDB é usado para dados intermediários serem armazenados, portanto, se os discos onde eles estão armazenados forem lentos, mais demorado será o processo de verificação;

· Complexidade do banco de dados: as mais diferentes características são colocadas no banco de dados, dessa forma, quanto mais funcionalidades são utilizadas no banco, mais tempo será necessário para executar as verificações de consistência;

· Opções utilizadas: as opções adicionais utilizadas no DBCC CHECKDB influenciam no tempo de execução do mesmo. Por exemplo, no uso do PHYSICAL_ONLY, o DBCC CHECKDB será executado mais rápido do que sem essa opção;

· Quantidade e tipos de corrupção: dependendo da quantidade e tipos de corrupção encontrados, pode ser necessário voltar e refazer alguma verificação para descobrir onde é a corrupção exatamente, levando mais tempo para conclusão.

Problemas do DBCC CHECKDB

Podem acontecer situações onde uma execução do DBCC CHECKDB começa a tomar muito tempo em comparação com as execuções anteriores. Isso provavelmente pode ser uma corrupção que foi encontrada e um algoritmo é acionado para identificar exatamente onde ela está. Alguns dos algoritmos que executam a verificação de consistência são projetados para serem executados e dizer se existe ou não uma corrupção da maneira mais rápida possível, mas quando uma corrupção acontece, ele não sabe onde foi. Sendo assim, deve ser feito um trabalho de voltar na verificação e efetuar uma análise profunda para descobrir exatamente onde ocorreu a corrupção.

Nesses casos em que a execução está demorando muito tempo, não encerre o processo do DBCC CHECKDB, deixe-o completar e lhe dizer que tipos de corrupção existem.

Outra situação envolve o TempDB que é utilizado pelo DBCC CHECKDB para armazenamento temporário de dados. Você deve garantir que o TempDB está dimensionado da forma correta. Caso o TempDB esteja sem espaço para crescer, as verificações de consistência irão falhar. Você pode usar a opção WITH ESTIMATEONLY para obter uma estimativa de quanto espaço será exigido do TempDB.

Sinais de corrupção

Existe uma variedade de sinais diferentes que indicam que uma corrupção está presente no banco de dados. Mesmo se você não estiver executando o DBCC CHECKDB, elas podem ser utilizadas como indícios de sua ocorrência:

· Usuários informando a quebra de conexão: provavelmente devido aos erros 823 e 824, a conexão desses usuários é quebrada quando eles estão tentando executar consultas ou a aplicação não está conseguindo se conectar corretamente;

· Jobs de backup com falha: as tarefas de backup podem começar a apresentar falhas, provavelmente devido ao erro 3043, que é retornado quando o backup detecta problemas no Checksum;

· Alerta dos erros 823 e 824: esses alertas podem começar a disparar mensagens de e-mail dizendo que erros estão acontecendo;

· Erros no Log de Erros: Uma análise no log de erros do SQL Server pode mostrar problemas que estão ocorrendo.

Executando o DBCC CHECKDB

A execução do DBCC CHECKDB é muito simples, basta utilizar o comando:

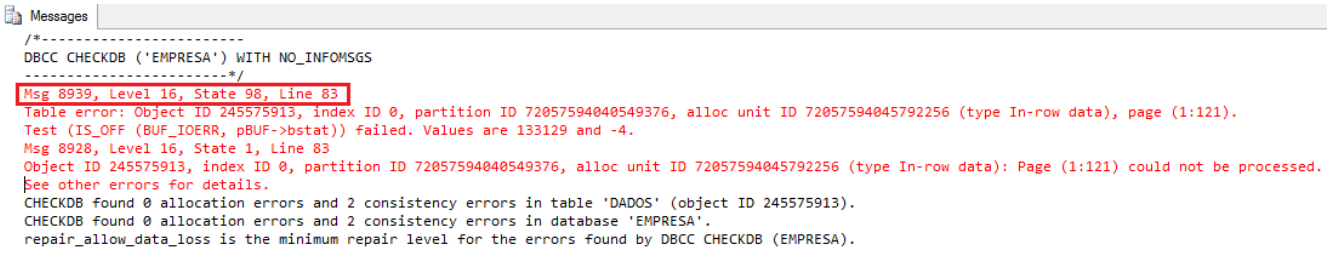
```
DBCC CHECKDB (‘Nome Banco de Dados’)
```

Caso você esteja utilizando o SQL Server 2008 RTM ou anterior, é necessário acrescentar também o comando ALL_ERRORMSGS para garantir que todas as mensagens de erro sejam retornadas.

Desde que o processo de execução do DBCC CHECKDB não seja interrompido de forma inesperada, será escrita uma mensagem de conclusão no log de erros do SQL Server e no log de eventos das aplicações do Windows. A saída regular só é apresentada para a conexão que está executando o comando.

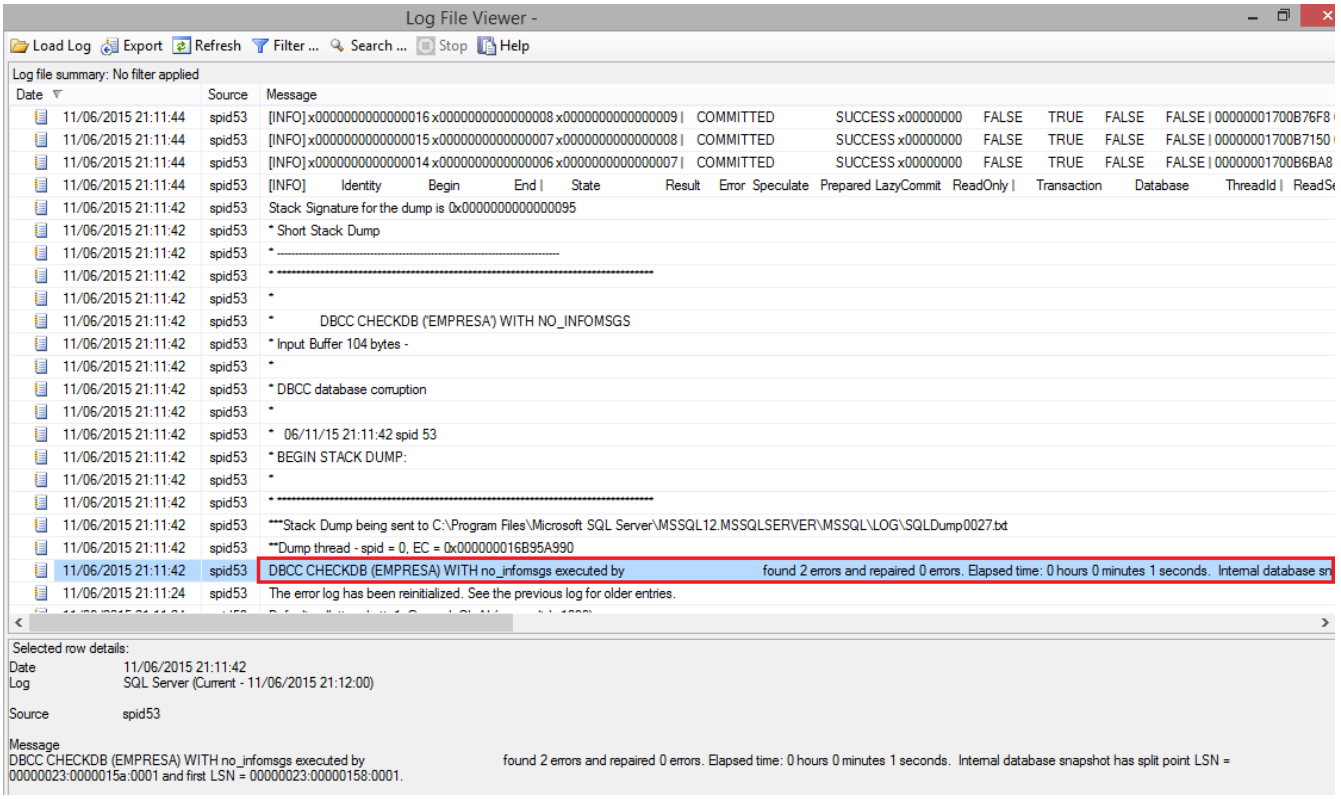
Caso uma corrupção seja encontrada, uma mensagem com severidade 16 fará parte do retorno.

Veja alguns exemplos nas **Figuras 5** a **7** de mensagens retornadas em sessões onde executamos o CHECKDB e problemas foram encontrados.



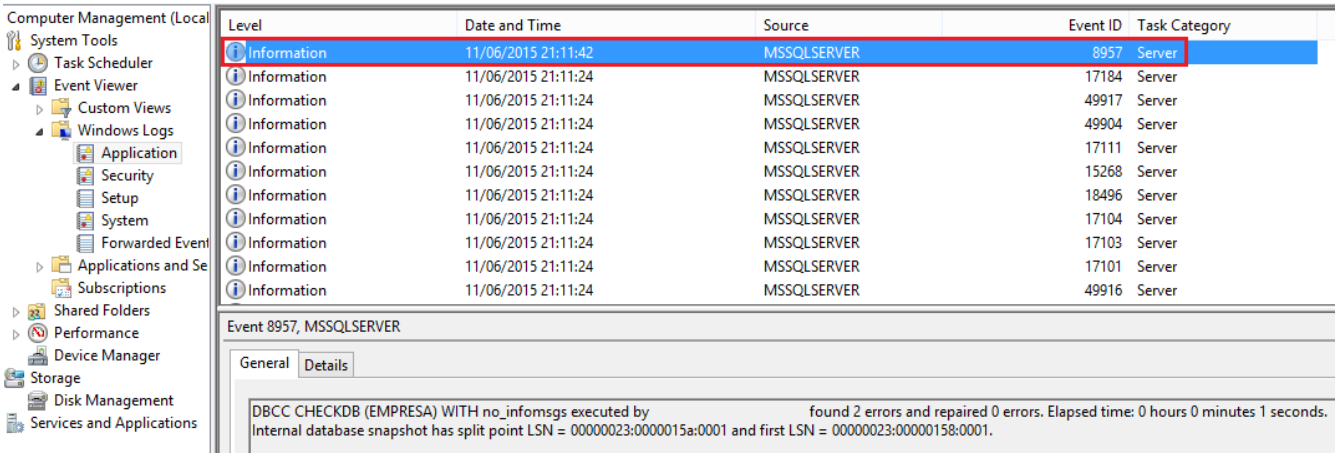
abrir imagem em nova janela

Figura 5. Retorno da mensagem de corrupção na sessão do usuário



abrir imagem em nova janela

Figura 6. Mensagem da corrupção no log de erro do SQL Server



abrir imagem em nova janela

Figura 7. Mensagem da corrupção no log de eventos do Windows

Há uma enorme variedade de erros que o DBCC CHECKDB pode produzir, alguns deles têm mais de 200 estados de mensagem diferentes, o que significa que existem mais de 1000 condições de corrupção distintas que podem ser relatadas no retorno da

verificação de consistência. Dessa forma, é muito difícil descobrir o significado de múltiplas corrupções quando estão combinadas, apesar de existirem algumas dicas que podem ajudar a decidir qual a melhor ação a ser tomada para cada situação. Veja algumas dicas:

· O DBCC CHECKDB falhou: pode ser que há algo errado dentro do banco de dados que o impede de ser executado, não sendo possível dessa forma utilizar os recursos de reparação. Caso isso aconteça, você realmente não tem escolha, a não ser restaurar a base a partir de um backup ou exportar os dados para um novo banco de dados;

· Erros fatais: são erros que impedem do DBCC CHECKDB concluir como, por exemplo, erros críticos nas tabelas de sistema, metadados corrompidos ou erros internos do próprio DBCC CHECKDB;

· Estado do erro: se o DBCC CHECKDB falhar, por qualquer motivo, a mensagem no log vai ter um estado de erro associado a ele e esses estados são documentados no Books Online da Microsoft;

· Corrupção apenas em índices NonClustered: pode ser verificada essa situação quando o nível de reparação recomendado é o Repair_Rebuild ou o ID do índice em todos os erros apresentados são maiores que 1, sendo assim, são apenas índices NonClustered com problemas. Nesse caso, não será necessária uma restauração ou reparação para corrigir esses índices, pois manualmente é possível efetuar uma reconstrução do índice sem ter que deixar o banco Offline;

· Erros irreparáveis: o DBCC CHECKDB só vai lhe dizer se o erro é irreparável ao tentar executar um reparo, mas se você já souber que eles existem, pode poupar algum tempo não tendo que recorrer à opção Repair_Allow_Data_Loss sabendo que não terá efeito sobre esses erros específicos. Alguns desses erros são:

o Erro 2570: erro de pureza de dados (Data Purity), em que o valor da coluna é inválido para o tipo de dado;

o Erro 8992: existe algum tipo de incompatibilidade de metadados;

o Erro 8909, 8938, 8939: erro na PFS (Page Free Space), que é um bitmap de alocação.

Nenhum desses tipos de erros podem ser reparados pelo DBCC CHECKDB, sendo assim, suas opções são restaurar do banco de dados, tentar exportar os dados para outra base ou reparar manualmente. Veja alguns exemplos de tipos comuns de corrupção nas **Listagens 10** a **12**. Para esses dois casos mostrados, o nível mínimo de reparo é Repair_Allow_Data_Loss, ou seja, a página que não pode ser lida por causa do erro de I/O, vai ser removida da alocação e a reparação vai ter que ajustar tudo que está ligado e gerar ligações a partir daquela página.

Na **Listagem 10** basicamente criamos a base de dados que será utilizada para testes. Nela também escolhemos a página de dados que será corrompida e inserimos o problema na base de dados.

Listagem 10. Exemplos dos tipos mais vistos de corrupção – Parte 1

```
USE MASTER
GO
IF DB_ID('EMPRESA') IS NOT NULL
ALTER DATABASE EMPRESA SET SINGLE_USER WITH ROLLBACK IMMEDIATE
GO
DROP DATABASE EMPRESA
GO
/*CRIE NOVAMENTE O BANCO DE DADOS EMPRESA E INSIRA ALGUNS VALORES NA TABELA*/
CREATE DATABASE EMPRESA ON PRIMARY (
 NAME = 'EMPRESA',
 FILENAME = 'C:\DEMO_CORRUPCAO\EMPRESA.MDF')
LOG ON (
 NAME = 'EMPRESA_LOG',
```

```
 FILENAME = 'C:\DEMO_CORRUPCAO\EMPRESA_LOG.LDF')
GO
USE EMPRESA
GO
CREATE TABLE DADOS(
 ID INT IDENTITY,
 VALOR CHAR (8000) DEFAULT 'A'
)
GO
INSERT INTO DADOS DEFAULT VALUES
GO 10
/*ESCOLHA UM PAGEPID DE UMA PÁGINA DE DADOS COM O PAGETYPE 10*/
DBCC IND ('EMPRESA', 'DADOS', -1)
GO
ALTER DATABASE EMPRESA SET SINGLE_USER

GO
/*A CORRUPÇÃO SERÁ FEITA EM UMA PÁGINA IAM*/
DBCC WRITEPAGE ('EMPRESA', 1, 118, 0, 2, 0x0000, 1)
GO
ALTER DATABASE EMPRESA SET MULTI_USER
GO
```

Já na **Listagem 11** criamos inicialmente um segundo banco e inserimos alguns valores para simular uma corrupção em uma página do índice. Feito isso, novamente escolhemos uma página de dados e corrompemos a base. Em seguida, efetuamos uma verificação de consistência para as duas bases de dados. Observe que temos vários erros e reparem que a página 118 não pode ser processada. Sempre que esse tipo de mensagem aparecer é uma boa indicação de algum erro de I/O. Olhando mais para o final, podemos encontrar a mensagem 8939 informando que há um erro de I/O e, logo mais acima, temos a informação que a página corrompida foi a IAM. Os erros 8906 informam que as páginas que estão alocadas ou marcadas como alocadas estão corrompidas.

Listagem 11. Exemplos dos tipos mais vistos de corrupção – Parte 2

```
CREATE DATABASE EMPRESA2 ON PRIMARY (
```

```
 NAME = 'EMPRESA2',
 FILENAME = 'C:\DEMO_CORRUPCAO\EMPRESA2.MDF')
LOG ON (
 NAME = 'EMPRESA2_LOG',
 FILENAME = 'C:\DEMO_CORRUPCAO\EMPRESA2_LOG.LDF')
GO
USE EMPRESA2
GO
CREATE TABLE DADOS(
 ID INT IDENTITY,
 VALOR CHAR (8000) DEFAULT 'A'
)
GO
INSERT INTO DADOS DEFAULT VALUES
GO 10
/*CRIE UM ÍNDICE CLUSTERED PARA TABELA*/
CREATE CLUSTERED INDEX IDX_DADOS_CLUSTERED
ON DADOS (ID)
GO
/*SELECIONE UMA PÁGINA QUE TENHA O PAGETYPE 2, QUE É UMA PÁGINA DE ÍNDICE*/
DBCC IND ('EMPRESA2', 'DADOS', -1)
GO
ALTER DATABASE EMPRESA2 SET SINGLE_USER
GO
DBCC WRITEPAGE ('EMPRESA2', 1, 150, 0, 2, 0x0000, 1)
GO
ALTER DATABASE EMPRESA2 SET MULTI_USER
GO
/*EXECUTE A VERIFICAÇÃO DE CONSISTÊNCIA PARA AMBAS AS BASES*/
DBCC CHECKDB ('EMPRESA') WITH NO_INFOMSGS
/*RESULTADO:
Msg 8928, Level 16, State 6, Line 114
Object ID 0, index ID -1, partition ID 0, alloc unit ID 0 (type Unknown): Page (1:
Msg 8905, Level 16, State 1, Line 114
Extent (1:232) in database ID 22 is marked allocated in the GAM, but no SGAM or IA
CHECKDB found 2 allocation errors and 0 consistency errors not associated with any
Msg 8939, Level 16, State 98, Line 114
Table error: Object ID 245575913, index ID 0, partition ID 72057594040549376, allc
Test (IS_OFF (BUF_IOERR, pBUF->bstat)) failed. Values are 133129 and -4.
Msg 8906, Level 16, State 1, Line 114
Page (1:94) in database ID 22 is allocated in the SGAM (1:3) and PFS (1:1), but wa
...
Msg 2575, Level 16, State 1, Line 114
The Index Allocation Map (IAM) page (1:118) is pointed to by the next pointer of I
alloc unit ID 72057594045792256 (type In-row data), but it was not detected in the
Msg 8939, Level 16, State 98, Line 114
```

```
Table error: Object ID 245575913, index ID 0, partition ID 72057594040549376, allc
Test (IS_OFF (BUF_IOERR, pBUF->bstat)) failed. Values are 133129 and -4.
Msg 7965, Level 16, State 2, Line 114
Table error: Could not check object ID 245575913, index ID 0, partition ID 720575
due to invalid allocation (IAM) page(s).
CHECKDB found 11 allocation errors and 2 consistency errors in table 'DADOS' (obje
CHECKDB found 13 allocation errors and 2 consistency errors in database 'EMPRESA'.
repair_allow_data_loss is the minimum repair level for the errors found by DBCC CH
GO
```

Por fim, na **Listagem 12** observamos que as mensagens estão diferentes para essa corrupção, mas temos a mensagem 8939 informando o erro no I/O na leitura da página 150. Em seguida temos vários erros de ligação na B-Tree, que são as mensagens 8977 informando que o nó pai da página não pode ser encontrado. Isso significa que existe uma página no meio do índice, com páginas filhas, que não puderam ser processadas por alguma razão.

Listagem 12. Exemplos dos tipos mais vistos de corrupção – Parte 3

```
DBCC CHECKDB ('EMPRESA2') WITH NO_INFOMSGS

/*RESULTADO:
Msg 8939, Level 16, State 98, Line 120
Table error: Object ID 245575913, index ID 1, partition ID 72057594040614912, allc
Test (IS_OFF (BUF_IOERR, pBUF->bstat)) failed. Values are 133129 and -4.
Msg 8928, Level 16, State 1, Line 120
Object ID 245575913, index ID 1, partition ID 72057594040614912, alloc unit ID 720
See other errors for details.
Msg 8979, Level 16, State 1, Line 120
Table error: Object ID 245575913, index ID 1, partition ID 72057594040614912, allc
Page (1:145) is missing references from parent (unknown) and previous (page (0:0))
Msg 8977, Level 16, State 1, Line 120
Table error: Object ID 245575913, index ID 1, partition ID 72057594040614912, allc
Parent node for page (1:147) was not encountered.
...
CHECKDB found 0 allocation errors and 13 consistency errors in table 'DADOS' (obje
CHECKDB found 0 allocation errors and 13 consistency errors in database 'EMPRESA2'
repair_allow_data_loss is the minimum repair level for the errors found by DBCC CH
```

Restauração vs Reparação

Muitos pensam que o reparo é o último recurso, mas nem sempre é. Pode ser que a restauração de um backup leve muito mais tempo do que uma reparação. Dependendo do negócio, pode-se considerar que o tempo de inatividade é mais importante que a perda de dados, dessa forma, a reparação com a opção Repair_Allow_Data_Loss pode ser a melhor escolha.

Não se esqueça, há sempre a opção de tentar exportar os dados do banco de dados danificado se a restauração ou reparação não for possível.

Para poder tomar a decisão entre restauração e reparação, existem alguns pontos que devem ser analisados como, por exemplo, se existem backups consistentes, se o DBCC CHECKDB apresentou falhas, se apenas os índices NonClustered que estão danificados ou se existem erros irreparáveis.

Recuperação pelo backup

Recuperação com o uso de backups é a melhor maneira de evitar a perda de dados, mas não é necessariamente a melhor maneira de se minimizar o tempo de inatividade. Esta alternativa depende totalmente dos tipos de backup que você tem e quando a corrupção ocorre.

Por exemplo, se você tem uma estratégia de backup onde todo domingo o backup Full é executado e durante a semana a cada uma hora é feito o backup de log, para poder se recuperar no domingo um pouco antes do backup full, muitos backups de log devem ser restaurados, o que pode levar um bom tempo para ser feito. Nesse cenário, o melhor seria ter backups full ou diferenciais diariamente, permitindo rápidas

restaurações.

Opções de restauração

Existem diferentes opções de restauração. Algumas delas são:

· With_Recovery: essa é a opção padrão e não permite que nenhuma outra restauração com arquivos de backup adicionais possa ser utilizada no processo para complementar a recuperação. Dessa forma, a base fica acessível aos usuários após a conclusão da restauração do backup full;

· With_NoRecovery: essa opção permite que múltiplos arquivos de backup possam ser utilizados para recuperação completa do banco de dados. Ao final da sequência de restauração deve ser utilizado o comando *Restore Database "Nome do Banco" With Recovery* para finalizar a sequência;

· With_Standby: essa opção é parecida com o With_NoRecovery, pois permite fazer a restauração através de múltiplos arquivos de backup. Entretanto, após a finalização da sequência, a base é criada como Read_Only.

Restauração em um ponto no tempo

Algumas vezes você pode não querer fazer a restauração até o ponto mais recente possível. Para esses casos, existe um recurso que permite selecionar um ponto específico no tempo para recuperação. Por exemplo, imagine um usuário que possa ter deletado algum dado e você deseja que o banco seja restaurado um pouco antes da deleção acontecer. O recurso que permite fazer a restauração em um ponto específico no tempo será extremamente útil para cenários como esse.

A utilização desse recurso deve iniciar com a restauração do mais recente backup full e

em seguida, o conjunto de backups de log também deve ser restaurado até que se chegue no ponto do tempo desejado. Para se certificar de que sua recuperação não vá além do ponto do tempo de interesse, você pode incluir a opção With Stopat, que inclusive é uma boa prática durante as operações de restauração. A opção With Stopat é geralmente usada com um valor de data e hora ou você pode utilizar um nome de transação, caso esteja trabalhando com transações marcadas. Veja nas **Listagens 13** a **15** exemplos de uso de algumas das várias opções deste tipo de restauração.

Na primeira listagem, criamos a base de dados que utilizaremos no exemplo, uma tabela e inserimos alguns valores exemplo. Em seguida, executamos um backup full da base de dados. Então, inserimos novos dados e depois, executamos um backup de log da base. Por fim, salvamos o horário entre os registros da quinta e sexta transação. Esse horário será usado para uma restauração em um ponto do tempo.

Listagem 13. Exemplo das opções de restauração – Parte 1

```
USE MASTER
GO
IF DB_ID('EMPRESA') IS NOT NULL
ALTER DATABASE EMPRESA SET SINGLE_USER WITH ROLLBACK IMMEDIATE
GO
DROP DATABASE EMPRESA
GO
/*CRIE NOVAMENTE O BANCO DE DADOS QUE UTILIZAMOS NOS EXEMPLOS*/
CREATE DATABASE EMPRESA ON PRIMARY (
 NAME = 'EMPRESA',
 FILENAME = 'C:\DEMO_CORRUPCAO\EMPRESA.MDF')
LOG ON (
 NAME = 'EMPRESA_LOG',
 FILENAME = 'C:\DEMO_CORRUPCAO\EMPRESA_LOG.LDF')
GO
USE EMPRESA
GO
/*CRIE UM TABELA PARA ARMAZENAR ALGUMAS TRANSAÇÕES*/
CREATE TABLE DADOS(
 ID INT IDENTITY,
 VALOR CHAR (8000) DEFAULT 'A'
```

```
)
GO
INSERT INTO DADOS VALUES
 ('PRIMEIRA TRANSACAO')
GO
/*FAÇA UM BACKUP FULL*/
BACKUP DATABASE EMPRESA
TO DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_FULL.BAK'
WITH INIT
GO
/*ADICIONE MAIS DADOS E FAÇA UM BACKUP DE LOG*/
INSERT INTO DADOS VALUES
 ('SEGUNDA TRANSACAO')
INSERT INTO DADOS VALUES
 ('TERCEIRO TRANSACAO')
GO
BACKUP LOG EMPRESA
TO DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_LOG1.BAK'
WITH INIT
GO
/*ADICIONE MAIS DADOS*/
INSERT INTO DADOS VALUES
 ('QUARTA TRANSACAO')
INSERT INTO DADOS VALUES
 ('QUINTA TRANSACAO')
GO
SELECT GETDATE()
--TEMPO - ?
GO
/*ADICIONE A SEXTA E SETIMA TRANSAÇÃO*/
INSERT INTO DADOS VALUES
 ('SEXTA TRANSACAO')
INSERT INTO DADOS VALUES
 ('SETIMA TRANSACAO')
GO
```

Já na **Listagem 14**, inicialmente realizamos um novo backup de log. Em seguida, simulamos um desastre que acaba com o banco de dados (drop database). Feito isso, demos início à restauração dos backups pelo full, seguido dos backups de log. Observe que uma mensagem de erro será retornada informando que o backup não pode ser restaurado porque não há arquivos prontos para reversão. Isso acontece devido ao

comportamento padrão da restauração que usa a opção WITH RECOVERY. Assim, iniciamos novamente a sequência de restauração, mas dessa vez com a opção NORECOVERY. Observe que agora concluímos a sequência de restauração com sucesso.

Listagem 14. Exemplo das opções de restauração – Parte 2

```
/*FAÇA OUTRO BACKUP DE LOG*/
BACKUP LOG EMPRESA
TO DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_LOG2.BAK'
WITH INIT
GO
/*SIMULAREMOS UM DESASTRE QUE ACABA COM O BANCO DE DADOS*/
USE MASTER
GO
DROP DATABASE EMPRESA
GO
/*INICIANDO PELO BACKUP FULL, FAÇA A RESTAURAÇÃO DOS TRÊS BACKUPS FEITOS ANTERIORM
RESTORE DATABASE EMPRESA
FROM DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_FULL.BAK'
WITH REPLACE
GO
/*AGORA OS BACKUPS DE LOG*/
RESTORE LOG EMPRESA
FROM DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_LOG1.BAK'
RESTORE LOG EMPRESA
FROM DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_LOG2.BAK'

RESTORE DATABASE EMPRESA
FROM DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_FULL.BAK'
WITH REPLACE, NORECOVERY
GO
/*AGORA OS BACKUPS DE LOG*/
RESTORE LOG EMPRESA
FROM DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_LOG1.BAK'
WITH NORECOVERY
RESTORE LOG EMPRESA
FROM DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_LOG2.BAK'
WITH NORECOVERY
GO
/*CONCLUIMOS A SEQUÊNCIA DE RESTAURAÇÃO COM SUCESSO, SENDO ASSIM, PODEMOS FINALIZA
RESTORE DATABASE EMPRESA WITH RECOVERY
```

```
GO
/*VERIFIQUE QUE TODOS OS DADOS FORAM RECUPERADOS*/
SELECT * FROM EMPRESA.DBO.DADOS
GO
```

Na **Listagem 15** inicialmente realizamos uma restauração com a opção STOPAT para podermos parar o processo em um determinado ponto do tempo. Esta opção só irá funcionar quando usada na restauração dos arquivos de log. Sendo assim, usamos o horário salvo na **Listagem 13** dentro da instrução. Observe que na mensagem de conclusão da restauração do log é informado que o backup contém registros que foram registrados antes do ponto no tempo que foi determinado, ou seja, o SQL Server detectou que você não precisava usar a opção WITH STOPAT nesse arquivo de log pois ele não contém as transações do ponto de tempo desejado. Ao final, executamos uma nova restauração e realizamos uma nova consulta para retornarmos os cinco registros recuperados.

Listagem 15. Exemplo das opções de restauração – Parte 3

```
/*VAMOS FAZER UMA RESTAURAÇÃO COM A OPÇÃO STOPAT PARA PODER PARAR EM UM DETERMINAD
RESTORE DATABASE EMPRESA
FROM DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_FULL.BAK'
WITH REPLACE, NORECOVERY
GO
RESTORE LOG EMPRESA
FROM DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_LOG1.BAK'
WITH STOPAT = '2015-06-15 20:55:19.170'
GO
/*FAÇA A RESTAURAÇÃO DO SEGUNDO BACKUP DE LOG*/
RESTORE LOG EMPRESA
FROM DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_LOG2.BAK'
WITH STOPAT = '2015-06-15 20:55:19.170'
GO
/*FAÇA UMA NOVA CONSULTA E VEJA SE APENAS OS PRIMEIROS 5 REGISTROS FORAM RECUPERAD
SELECT * FROM EMPRESA.DBO.DADOS
```

Tail-Log Backup

Quando um desastre ocorre e você vai restaurar a partir dos arquivos backups, deve ser garantido que será pego a parte final do log de transação, que ainda não foi tirado o backup, permitindo a restauração até o ponto no tempo em que o incidente ocorreu, sem perda de dados. Essa cópia de segurança especial é chamada de tail-log backup.

Caso você esteja tentando fazer um tail-log backup e não é possível devido aos arquivos de dados estarem inacessíveis ou danificados, você pode forçar a execução do tail-log backup usando a opção With No_Truncate. Nas **Listagens 16 e 17**, temos um exemplo de uso do tail-log backup. Nelas, simulamos um desastre e colocamos o banco de dados offline para deletarmos o arquivo de dados. Em seguida, tentamos colocar a base online novamente, mas percebemos que um erro de localização do arquivo de dados não permitirá que isto ocorra. Feito isso, fazemos a restauração do banco com os arquivos de backup, mas com um nome diferente e em diretórios distintos para analisarmos o que temos de dados. Após uma série de restaurações de arquivos de log, consultamos novamente a base de dados e percebemos que todos os registros foram recuperados com sucesso. Mesmo aqueles que não estavam no backup feito antes do desastre. Dessa forma, percebemos que eles foram salvos pelo tail-log backup e foi possível adicionar esse backup ao final da sequência de restauração garantindo que não houvesse nenhuma perda de dados.

Listagem 16. Exemplo da aplicação do tail-log backup – Parte 1

```
USE MASTER
GO
IF DB_ID('EMPRESA') IS NOT NULL
ALTER DATABASE EMPRESA SET SINGLE_USER WITH ROLLBACK IMMEDIATE
GO
DROP DATABASE EMPRESA
GO
/*CRIE O BANCO DE DADOS EMPRESA NOVAMENTE*/
```

```
CREATE DATABASE EMPRESA ON PRIMARY (
 NAME = 'EMPRESA',
 FILENAME = 'C:\DEMO_CORRUPCAO\EMPRESA.MDF')
LOG ON (
 NAME = 'EMPRESA_LOG',
 FILENAME = 'C:\DEMO_CORRUPCAO\EMPRESA_LOG.LDF')
GO
USE EMPRESA
GO
CREATE TABLE DADOS(
 ID INT IDENTITY,
 VALOR CHAR (8000) DEFAULT 'A'
)
GO
/*INSIRA O DADO A SEGUIR E FAÇA UM BACKUP FULL*/
INSERT INTO DADOS VALUES
 ('PRIMEIRA TRANSACAO')
GO
BACKUP DATABASE EMPRESA
TO DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_FULL.BAK'
WITH INIT
GO
/*ADICIONE MAIS DADOS*/
INSERT INTO DADOS VALUES
 ('SEGUNDA TRANSACAO')
GO
INSERT INTO DADOS VALUES
 ('TERCEIRA TRANSACAO')
GO
/*FAÇA UM BACKUP DE LOG*/
BACKUP LOG EMPRESA
TO DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_LOG1.BAK'
WITH INIT
GO
/*ADICIONE MAIS DADOS*/
INSERT INTO DADOS VALUES
 ('QUARTA TRANSACAO')
GO
INSERT INTO DADOS VALUES
 ('QUINTA TRANSACAO')
GO
```

Listagem 17. Exemplo da aplicação do tail-log backup – Parte 2

```
--SIMULE UM DESASTRE, COLOQUE A O BANCO DE DADOS OFFLINE E DELETE O ARQUIVO DE DAD
ALTER DATABASE EMPRESA SET OFFLINE
GO
/*TENTE COLOCAR A BASE ONLINE NOVAMENTE, MAS VEJA QUE UM ERRO DE LOCALIZAÇÃO DO AR
ALTER DATABASE EMPRESA SET ONLINE
GO
/* AGORA VAMOS FAZER A RESTAURAÇÃO DO BANCO COM OS ARQUIVOS DE BACKUP, MAS COM UM
RESTORE DATABASE EMPRESA_COPIA
FROM DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_FULL.BAK'
WITH MOVE 'EMPRESA' TO 'C:\DEMO_CORRUPCAO\EMPRESA_COPIA.MDF',
     MOVE 'EMPRESA_LOG' TO 'C:\DEMO_CORRUPCAO\EMPRESA_COPIA_LOG.MDF',
     REPLACE, NORECOVERY
GO
/*FAÇA A RESTAURAÇÃO DO ARQUIVO DE LOG*/
RESTORE LOG EMPRESA_COPIA
FROM DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_LOG1.BAK'
WITH NORECOVERY
GO
/*FINALIZE A SEQUÊNCIA DE RESTAURAÇÃO*/
RESTORE DATABASE EMPRESA_COPIA WITH RECOVERY
GO
/*REPARE QUE PERDEMOS OS 2 ÚLTIMOS REGISTROS POR QUE ELES ESTÃO NO TAIL-LOG E AIND
SELECT * FROM EMPRESA_COPIA.DBO.DADOS
GO
/*TENTE FAZER O TAIL-LOG BACKUP, MAS VEJA QUE NÃO SERÁ POSSÍVEL POR QUE O ARQUIVO
BACKUP LOG EMPRESA
TO DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_LOG_TAIL.BAK'
WITH INIT
GO
/*PARA CONTORNAR ESSA RESTRIÇÃO USAREMOS A OPÇÃO WITH NO_TRUNCATE, QUE FARÁ O BACK
BACKUP LOG EMPRESA
TO DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_LOG_TAIL.BAK'
WITH INIT, NO_TRUNCATE
/*INICIE NOVAMENTE A SEQUÊNCIA DE RESTAURAÇÃO NA BASE EMPRESA_COPIA*/
RESTORE DATABASE EMPRESA_COPIA
FROM DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_FULL.BAK'
WITH MOVE 'EMPRESA' TO 'C:\DEMO_CORRUPCAO\EMPRESA_COPIA.MDF',
     MOVE 'EMPRESA_LOG' TO 'C:\DEMO_CORRUPCAO\EMPRESA_COPIA_LOG.MDF',
     REPLACE, NORECOVERY
GO
/*RESTAURE O BACKUP DE LOG INICIAL*/
RESTORE LOG EMPRESA_COPIA
FROM DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_LOG1.BAK'
WITH NORECOVERY
GO
/*RESTAURE O BACKUP DO TAIL-LOG*/
```

```
RESTORE LOG EMPRESA_COPIA
FROM DISK = 'C:\DEMO_CORRUPCAO\EMPRESA_LOG_TAIL.BAK'
WITH NORECOVERY
GO
/*FINALIZE A SEQUÊNCIA DE RESTAURAÇÃO*/
RESTORE DATABASE EMPRESA_COPIA WITH RECOVERY
GO
SELECT * FROM EMPRESA_COPIA.DBO.DADOS
```

Sempre que ocorrer um desastre, nunca se esqueça de primeiramente fazer o tail-log backup e lembre-se, assim que iniciar o processo de restauração do banco de dados, a capacidade de tirar um tail-log backup é perdida.

Estratégias de reparação

Na maioria das vezes em que o reparo vai ser utilizado, é porque você não tem os backups necessários para fazer a restauração sem a perda de dados. É bem improvável, a menos que esteja descrito no seu SLA (*Service Level Agreement*), que o tempo de inatividade seja mais importante que a perda de dados para que você tenha que usar o reparo como primeira opção.

Deve-se ter muito cuidado quando estiver usando o reparo ou quando estiver corrigindo as corrupções manualmente, porque esta ação pode envolver perda de dados e não queremos piorar a situação. Isso significa que você vai ter que praticar o uso do reparo para se certificar de que sabe o que vai fazer.

Funcionamento da reparação

O propósito da reparação é fazer com que o banco de dados fique estruturalmente consistente, de modo que o mecanismo de armazenamento possa processar o banco sem corrupções. Essa tarefa é feita o mais rápido possível, razão pela qual a maioria

dos reparos, como a opção Repair_Allow_Data_Loss, existem para apagar e arrumar o que está danificado.

Para saber o que deve ser reparado, cada etapa da execução do DBCC CHECKDB verifica e processa uma lista de corrupções. A decisão do que deve ser reparado primeiramente é baseada em um ranking existente para todas as corrupções, onde é determinado o quão intrusiva a reparação vai ser para ela e a mais intrusiva é processada primeiro.

O reparo completo nem sempre pode ser feito, sendo necessário verificar a saída da execução do DBCC CHECK, pois lá são mostrados todos os erros e suas correções, caso tenha sido feita com sucesso ou não.

Não importa qual tipo de opção de reparo for utilizada, todas requerem que o banco esteja em Single_User, ou seja, seu banco de dados vai estar inacessível enquanto o reparo estiver sendo executado, pois não é possível fazer esse trabalho com o banco online tendo modificações nos dados ao mesmo tempo que o processo é executado.

É possível iniciar uma transação antes da operação de recuperação. Você pode executar o Repair_Allow_Data_Loss dentro da transação e, caso não goste do resultado, pode reverter através do rollback.

Opções de reparação

Para saber qual opção de reparação usar, você deve concluir a execução do DBCC CHECKDB e ele irá lhe dizer qual opção de reparo é necessário ao final das mensagens de erro. Veja as opções existentes:

· Rapair_Fast: essa opção não faz nada e existe apenas para compatibilidade de versões anteriores ao SQL Server 2005;

· Repair_Rebuild: irá executar reparos que não causam qualquer perda de dados, ou seja, basicamente a reconstrução de índices NonClustered danificados;

· Repair_Allow_Data_Loss: na maioria das vezes, é esse comando que será usado. Os reparos feitos estão suscetíveis a perda de dados, mas não sempre.

Repair_Allow_Data_Loss

Esse nome é dado a essa opção de recuperação porque, geralmente, corrige os problemas de inconsistência removendo a alocação e consertando todas as ligações. Esse processo é o mais rápido e correto para efetuar os reparos.

O Repair_Allow_Data_Loss não leva em conta restrições de chave estrangeira, qualquer lógica de negócio herdada e replicações. Depois de executar o reparo, certifique-se de verificar os dados para ver o que foi perdido e execute novamente o DBCC CHECKDB para ter certeza de que todas as corrupções foram reparadas. Se necessário, execute o DBCC CHECKCONSTRAINT nas tabelas que foram reparadas para garantir que as constraints estejam válidas. Toda vez que alguma tabela da replicação for reparada, você deve reinicializar qualquer uma das topologias de replicação envolvidas. Veja exemplos da execução do reparo em alguns cenários comuns nas **Listagens 18 e 19**. Na primeira listagem, basicamente preparamos a base de dados que será utilizada nos exemplos. Nela criamos uma base contendo uma tabela. Inserimos valores de exemplo e construímos um índice. Por fim, corrompemos a página do índice criado.

Listagem 18. Exemplos de reparos – Parte 1

```
USE MASTER
GO
IF DB_ID('EMPRESA') IS NOT NULL
ALTER DATABASE EMPRESA SET SINGLE_USER WITH ROLLBACK IMMEDIATE
```

```
GO
DROP DATABASE EMPRESA
GO
IF DB_ID('EMPRESA2') IS NOT NULL
ALTER DATABASE EMPRESA2 SET SINGLE_USER WITH ROLLBACK IMMEDIATE
GO
DROP DATABASE EMPRESA2
GO
/*CRIE NOVAMENTE O BANCO EMPRESA PARA SIMULAR UMA CORRUPÇÃO NA PÁGINA IAM*/
CREATE DATABASE EMPRESA ON PRIMARY (
 NAME = 'EMPRESA',
 FILENAME = 'C:\DEMO_CORRUPCAO\EMPRESA.MDF')
LOG ON (
 NAME = 'EMPRESA_LOG',
 FILENAME = 'C:\DEMO_CORRUPCAO\EMPRESA_LOG.LDF')
GO
USE EMPRESA
GO
CREATE TABLE DADOS(
 ID INT IDENTITY,
 VALOR CHAR (8000) DEFAULT 'A'
)
GO
INSERT INTO DADOS DEFAULT VALUES
GO 10
/*LISTE AS PÁGINAS DA TABELA E SELECIONE O PAGEPID ONDE O PAGETYPE FOR IGUAL A 10*
DBCC IND ('EMPRESA','DADOS', -1)
GO
ALTER DATABASE EMPRESA SET SINGLE_USER
GO
/*CAUSE A CORRUPÇÃO*/
DBCC WRITEPAGE ('EMPRESA', 1, 118, 0, 2, 0x0000, 1)
GO
ALTER DATABASE EMPRESA SET MULTI_USER
GO
/*PARA CORROMPER UMA PÁGINA DO ÍNDICE CRIE O BANCO EMPRESA2*/
CREATE DATABASE EMPRESA2 ON PRIMARY (
 NAME = 'EMPRESA2',
 FILENAME = 'C:\DEMO_CORRUPCAO\EMPRESA2.MDF')
LOG ON (
 NAME = 'EMPRESA2_LOG',
 FILENAME = 'C:\DEMO_CORRUPCAO\EMPRESA2_LOG.LDF')
GO
USE EMPRESA2
GO
CREATE TABLE DADOS(
```

```
 ID INT IDENTITY,
 VALOR CHAR (8000) DEFAULT 'A'
)
GO
INSERT INTO DADOS DEFAULT VALUES
GO 10
/*CRIE UM CLUSTER ÍNDICE*/
CREATE CLUSTERED INDEX DADOS_CLUSTERED
ON DADOS (ID)
GO
/*SELECIONE UM PÁGINA QUE TENHA UM PAGETYPE IGUAL A 2*/
DBCC IND ('EMPRESA2','DADOS', -1)
GO
ALTER DATABASE EMPRESA2 SET SINGLE_USER
GO
```

Na **Listagem 19** inicialmente causamos uma nova corrupção na base de dados para então rodarmos o comando DBCC CHECKDB. O erro será apresentado como esperado. Note que no rodapé da mensagem é informado que o nível mínimo de recuperação é o REPAIR_ALLOW_DATA_LOSS. Vamos utilizar justamente esta opção de recuperação. Para isso, inicialmente colocamos a base como SINGLE_USER. Após a execução do reparo, as páginas serão corrigidas para o índice cluster e as alocações necessárias serão refeitas, basicamente recriando aquela página corrompida.

Sempre que uma reparação for feita, execute novamente o DBCC CKECKDB para garantir que não existem outras corrupções no banco. Em seguida, coloque a base em MULTI_USER novamente caso não existam novos erros a serem reparados. Para esse exemplo, nenhum dado foi perdido porque não havia nada a ser eliminado no processo de reparação.

Listagem 19. Exemplos de reparos – Parte 2

```
/*CAUSE A CORRUPÇÃO*/
DBCC WRITEPAGE ('EMPRESA2', 1, 180, 0, 2, 0x0000, 1)
GO
```

```
ALTER DATABASE EMPRESA2 SET MULTI_USER
GO
/*EXECUTE O DBCC CHECKDB PARA PRIMEIRA BASE COM O CORRMPIMENTO DA PÁGINA IAM*/
DBCC CHECKDB ('EMPRESA') WITH NO_INFOMSGS
GO
ALTER DATABASE EMPRESA SET SINGLE_USER
GO
DBCC CHECKDB ('EMPRESA', REPAIR_ALLOW_DATA_LOSS)
WITH NO_INFOMSGS
GO
DBCC CHECKDB ('EMPRESA') WITH NO_INFOMSGS
GO
ALTER DATABASE EMPRESA SET MULTI_USER
GO
/*EXECUTE OS MESMOS PASSOS PARA SEGUNDA BASE*/
/*NESSA CORRUPÇÃO AS MENSAGEM APRESENTADAS INFORMARAM SOBRE A QUEBRA DAS LIGAÇOES
NOVAMENTE O NÍVEL MÍNIMO DE REPARO É O REPAIR_ALLOW_DATA_LOSS.*/
DBCC CHECKDB ('EMPRESA2') WITH NO_INFOMSGS
GO
ALTER DATABASE EMPRESA2 SET SINGLE_USER
GO
/*A MENSAGEM APRESENTADA SERÁ DA CONCLUSÃO DA RETIRADA DA ALOCAÇÃO DO ÍNDICE CLUST
DBCC CHECKDB ('EMPRESA2', REPAIR_ALLOW_DATA_LOSS)
WITH NO_INFOMSGS
GO
/*EXECUTE NOVAMENTE PARA GARANTIR QUE NÃO SOBRARAM ERROS*/
DBCC CHECKDB ('EMPRESA2') WITH NO_INFOMSGS
GO
ALTER DATABASE EMPRESA2 SET MULTI_USER
/*EM AMBOS OS CASOS NENHUM DADO FOI PERDIDO, POIS NADA TEVE QUE SER DELETADO PARA
```

Corrigindo índices nonclustered

Qualquer tipo de reparo requer que o banco de dados esteja offline, ou seja, em modo Single_User. Dessa forma, não faz muito sentido corrigir índices NonClustered usando o CHECKDB ou CHECKTABLE com a opção Repair_Rebuild, porque o banco precisa estar offline.

O índice apenas precisa ser desativado e reconstruído, o que pode ser feito

manualmente mantendo o banco online. Deve-se ficar atento para não executar somente a reconstrução do índice sem desativá-lo, dessa forma, o antigo índice será lido para construir um novo.

É recomendado que as operações sejam feitas dentro de uma transação, deve ser usado o comando ALTER INDEX para desabilitar e em seguida reconstruir o índice. A transação irá garantir que não ocorram violações de constraint durante esse processo. Observe na **Listagem 20** um exemplo de como fazer a reparação em índices NonClustered. Observe que alguns poucos erros foram retornados devido ao pequeno volume de corrupção que foi simulado. Entretanto, em um cenário real o resultado da verificação pode ser muito maior, gerando certa dificuldade para a análise. É possível visualizar o resultado de forma tabular. Para isso, devemos utilizar a opção TABLERESULTS como complemento do DBCC CHECKDB. Ao analisar os resultados, observe algumas informações importantes como o INDEXID igual a 2 indicando que é algo relacionado ao índice, o MESSAGE TEXT indicando em qual tabela o erro ocorreu, e o REPAIRLEVEL que informa o nível de reparo mínimo para cada corrupção.

Listagem 20. Reparação em índices NonClustered

```
USE MASTER
GO
IF DB_ID('EMPRESA') IS NOT NULL
ALTER DATABASE EMPRESA SET SINGLE_USER WITH ROLLBACK IMMEDIATE
GO
DROP DATABASE EMPRESA
GO
/*CRIE O BANCO DE DADOS EMPRESA E INSIRA ALGUNS DADOS*/
CREATE DATABASE EMPRESA ON PRIMARY (
 NAME = 'EMPRESA',
 FILENAME = 'C:\DEMO_CORRUPCAO\EMPRESA.MDF')
LOG ON (
 NAME = 'EMPRESA_LOG',
 FILENAME = 'C:\DEMO_CORRUPCAO\EMPRESA_LOG.LDF')
GO
```

```
USE EMPRESA
GO
CREATE TABLE DADOS(
 ID INT IDENTITY,
 VALOR CHAR (8000) DEFAULT 'A'
)
GO
INSERT INTO DADOS DEFAULT VALUES
GO 1000
/*CRIE O ÍNDICE NONCLUSTERED QUE SERÁ CORROMPIDO*/
CREATE NONCLUSTERED INDEX DADOS_NONCLUSTERED
ON DADOS (ID)
/*CONSULTE A TABELA DE SISTEMA PARA PEGAR O ID DO ÍNDICE QUE CRIAMOS*/
SELECT INDEX_ID FROM SYS.INDEXES
WHERE NAME = 'DADOS_NONCLUSTERED'
AND OBJECT_ID = OBJECT_ID ('DADOS')
GO
/*LISTE AS PÁGINAS DO ÍNDICE E SELECIONE O PAGEPID PARA DA PRIMEIRA PÁGINA COM PAG
DBCC IND ('EMPRESA', 'DADOS', 2)
GO
ALTER DATABASE EMPRESA SET SINGLE_USER
GO
/*CORROMPA ESSE REGISTRO*/
dbcc writepage ('EMPRESA', 1, 145, 134, 1, 0x64);
GO
ALTER DATABASE EMPRESA SET MULTI_USER
GO
DBCC CHECKDB ('EMPRESA') WITH NO_INFOMSGS
GO
DBCC CHECKDB ('EMPRESA') WITH NO_INFOMSGS, TABLERESULTS
GO
/*VAMOS FAZER A CORREÇÃO MANUALMENTE DO ÍNDICE AFETADO*/
USE EMPRESA
GO
/*ENCONTRE O NOME DO ÍNDICE NONCLUSTERED COM ERRO*/
EXEC SP_HELPINDEX 'DADOS'
GO
/*DESABILITE O ÍNDICE E FAÇA O REBUILD DENTRO DE UMA TRANSAÇÃO PARA GARANTIR A CON
BEGIN TRAN
GO
ALTER INDEX DADOS_NONCLUSTERED ON DADOS DISABLE;
ALTER INDEX DADOS_NONCLUSTERED ON DADOS REBUILD
GO
COMMIT TRAN
GO
/*FAÇA NOVAMENTE A VERIFICAÇÃO*/
```

```
DBCC CHECKDB ('EMPRESA') WITH NO_INFOMSGS
```

Pureza dos dados

Quando o SQL Server encontra um erro 2570 de pureza de dados, não é possível repará-lo. Esse erro informa que o valor da coluna está fora dos limites para o tipo de dado utilizado. O SQL Server não consegue reparar esse erro porque o valor para correção da coluna não pode ser escolhido automaticamente e, se fosse possível, influenciaria diretamente na lógica do negócio sem uma avaliação prévia, sendo assim, devemos corrigir esse erro manualmente. O valor incorreto da coluna pode ser encontrado através de uma consulta na tabela ou utilizando o comando DBCC PAGE. Em seguida, aquela coluna deve ser atualizada para um valor válido.

Corrupções acontecem a todo momento e é bom estar preparado para quando acontecer. Sendo assim, quanto mais você praticar, mais fácil vai ser para se recuperar quando uma corrupção acontecer realmente em seu ambiente.

Não importa o método utilizado para corrigir o problema, você sempre deve saber a causa da corrupção para que possa tomar medidas que evitem futuros problemas. Para isso, execute diagnósticos no hardware, como testes no subsistema de I/O e na memória. Além disso, pesquise os erros apresentados no log do SQL Server em fóruns especializados e suporte. Verifique também o log de eventos do Windows em busca de pistas específicas. Se possível, entre em contato com a equipe de storage para obter possíveis alertas gravados em seus softwares.

Bruno Silva

Bruno Silva é Graduado em Ciência da Computação pelo UniCEUB - Centro Universitário de Brasília. Participante ativo da comunidade SQL Server DF, dos eventos da Microsoft e seus parceiros. Possui as certificações MCP e MCSA SQL S [...]

Publicado em 2015

O que você achou deste post?

Gostei (1) (0)

+ Mais conteúdo sobre SQL

Não há comentários

Postar dúvida / Comentário

Meus comentarios

Mais posts

Video aula

Agrupando registros com Group by - Curso Completo MySQL - Aula 39

Video aula

Busca redundante em uma tabela - Curso Completo MySQL - Aula 38

Video aula

Obtendo dados de mais de uma tabela - Curso Completo MySQL - Aula 37

Video aula

Introdução ao Comando Select - Curso Completo MySQL - Aula 36

Video aula

Entendendo os tipos de dados Data no MySQL - Curso Completo MySQL - Aula 35

Aplicativo com fontes

Código Fonte - Curso Dominando XML com SQL Server

Listar mais conteúdo