stellar

MICROSOFT SQL DATABASE CORRUPTION

TROUBLESHOOTING GUIDE

Get insights into SQL database integrity and consistency issues & how to fix database corruption in different scenarios.

CONTENTS

INTRODUCTION

Microsoft SQL Server is a Relational Database Management System (RDBMS) used to manage vast data. It supports a broad range of dataset operations, including data storage and retrieval, business analytics and BI, transaction processing, etc. An extensive "enterprise-grade" SQL database might handle several Terabytes of business-critical information, customer data, etc., enabling a seamless frontend experience by processing several thousand queries in seconds.

Maintaining this enormous database's integrity and availability is a crucial responsibility for the Database Administrator (DBA) and SQL database corruption poses a formidable challenge to the fulfillment of this responsibility. Firstly, SQL database corruption may happen gradually without sufficient indications to allow early detection. Therefore, the Administrator may discover SQL corruption only after it begins affecting the frontend operations, leaving no recourse unless there is a restorable backup. Also, the large size of SQL databases complicates troubleshooting SQL database corruption issues.

This Guide shares detailed information on SQL Server database corruption, including aspects like what is SQL database corruption, types, scenarios, & causes, & how to fix SQL database corruption. Our purpose is to provide a consolidated and credible reference with actionable insights to support the troubleshooting of SQL database corruption. The information covered herein applies to SQL Server 2019, 2017, 2016, 2014, and earlier versions.

WHAT IS SQL DATABASE CORRUPTION?

Understanding the nature of SQL database corruption becomes easy with the awareness of where and how the data is stored in a SQL database. There are different database structures and units that store the data in a SQL database, as follows:

1. **Record:** This is essentially a row for storing the individual data records comprising table data, indexes, metadata, boot structures, etc. Each data record stores the "data type" information, i.e., fixed-length or variable length, record type, record length, columns with null values, etc.

2. **Page:** Pages are the smallest unit of data for storing the data records, index records, and metadata (stored in Boot page). The structure of a SQL database page comprises Header and Body, totaling 8 KB in size. The header consists of information such as the available free space and total records in the page body, the object associated with the page, preceding and succeeding pages, etc. The page body consists of a record offset array at the end, which contains the pointers to the records' physical storage location (byte index). **Any mutation or defect in the page header or body, including the slot array, can affect the database integrity, causing page-level corruption in the SQL database.**

3. **Heap:** A heap is essentially a table without a clustered index. Microsoft SQL Server relies on Index Allocation Map (IAM) — a type of page for keeping track of heaps and indexes in the same database. A single IAM can track about 4 GB of data in groups of eight pages called extents. The IAM page Header section stores the address to the first extent in the series of mapped pages up to the size limit of 4 GB, beyond which SQL Server creates another **IAM page referenced through a pointer in the first IAM page. IAM page issues or an alteration in the chain of connections is another way to perceive SQL database corruption.**

4. **Index:** An index is a data structure that stores the data pages in a logical sequence, irrespective of their physical storage sequence, based on the keys in a B-Tree structure. The purpose of an index is to retrieve a row (or record) from the table or view efficiently. There are two types of indexes, namely clustered index and non-clustered index, as follows:

- **Clustered Index:** stores the b-tree, keys, and the actual record at the leaf level
- **Non-Clustered Index:** stores the b-tree, keys, and pointers (addresses) to the rows

SQL Server reads the database records by scanning the b-tree through the pointers connecting the different pages in the tree. These pointers are stored in sys.sysallocunits – a System Base Tables used to store the metadata information (here, pointer) of each storage allocation unit. **An anomaly in the index data structure can affect the database's ability to read and retrieve the database records, especially in non-clustered indexes. This situation is considered as SQL database corruption.**

CAUSES OF SQL DATABASE CORRUPTION

As understood, the root cause of SQL database corruption is a mutation or defect in the database page or data structures like heap and index. Broadly, there are three reasons for corruption, namely—

1. HARDWARE FAILURE OR CRASH

SQL Server runs on a computer system and stores the database files on storage disks, which could be individual hard disk drives configured on a rack-mount server or a Redundant Array of Independent Disks (RAID). The storage hardware and computer can deteriorate, fail, and crash due to reasons such as the follows —

- Mechanical wear and tear
- Disk subsystem issues
- Power surge or failure
- Motherboard failure

These hardware-related reasons can lead to SQL database corruption due to abrupt stoppage or crashing of the database, bad sectors on the storage disk, disk subsystem failure, etc. Notably, hardware failure is a significant reason behind SQL database corruption.

2. SOFTWARE ISSUES

There are software-related (or logical) issues resulting in SQL database corruption. The most common of these include—

- Improper SQL Server version upgrade
- Malware infection
- Firmware bug
- Incompatible or faulty drivers

TYPES OF SQL DATABASE CORRUPTION

SQL Server database corruption types can manifest at various levels (pages, log file, etc.) and in several forms, as this section outlines.

1. SQL PAGE LEVEL CORRUPTION

In a previous section, we understood that a SQL database page is the smallest unit of data for storing the records, indexes, etc. SQL Server stores the data and schema in a specific type of page called the MDF file (.mdf file extension), also known as the primary database file. This file comprises a header, body, and slot array like any other SQL Server database page. Any alteration or removal of information in these sections results in a SQL database page level corruption.

2. BOOT PAGE CORRUPTION

There is a boot page in the SQL Server database, storing the metadata such as database version, database name and ID, compatibility, checkpoint Log Sequence Number (LSN), purity status, etc. When a SQL Server instance is initiated, it reads the boot page and records the last CHECKDB run date in the error log. Any defect in the boot page hampers this start-up process and affects the recovery of a crashed or corrupted SQL database.

This is because it is impossible to perform a page-level restoration for a corrupted boot page or fix it using DBCC CHECKDB. Troubleshooting the boot page corruption scenarios is inherently challenging considering there is only one boot page per database, i.e., page 9 in the first data file, and boot page issues can hamper the backup process.

3. INDEX CORRUPTION

In a SQL database, indexes are the data structures used for retrieving table records. As learned previously, there can be clustered or non-clustered indexes based on how they store the B-Tree, keys, database records, and pointers. Any alteration or defect in how this information is organized in the index can affect the database's ability to store and retrieve the records, which is known as index corruption.

4. SQL DATABASE IN SUSPECT MODE

SQL Server can mark a database "Suspect" typically when it fails to recover the database to a consistent transactional state due to corruption in the form of torn pages, etc. The root cause this issue could be I/O problems, disk issues, or other underlying hardware-related abnormalities. The Suspect Mode situation arrives while initiating a SQL Server instance, attaching a database, or restoring the database or transaction logs.

SQL DATABASE CORRUPTION — COMMON ERROR CODES

SQL Server database engine generates specific error codes to indicate the corruption instances. Understanding these error messages helps in troubleshooting these issues. The following table summarizes these error codes:

#	Error Code	Error Type	Description	Severity Level
1.	**SQL error 823**	Operating system-level issue	The operating system returns error code 823 when Windows API calls fail to perform file Input/ output operations.	High — error 823 can affect database integrity & therefore requires immediate rectification.
2.	**SQL error 824**	Page-level error	SQL data engine returns error code 824 when a logical consistency check fails after reading or writing a page.	High — error 824 indicates a hardware or storage system problem resulting in file system issues or database corruption.
3.	**SQL error 825**	Hardware issue	Error 825 occurs when SQL needs to reissue the read operation, indicating a major issue with the disk hardware.	High — requires immediate resolution to avoid database corruption and data loss.
4.	**SQL error 605**	Page-level issue	Error 605 occurs when the SQL server encounters corruption while reading the pages from a table.	High — error 605 is considered high severity if it doesn't resolve itself (i.e., non-transient)
5.	**SQL error 926**	Database-level issue	SQL database is marked "Suspect Mode" when it fails to recover the database to a consistent state.	High — can hamper business continuity if it remains unresolved for a prolonged duration.
6.	**SQL error 963**	Database-level issue	SQL database is in suspect mode due to events that happened during the upgrade	Moderate — requires action to restore the database online.
7.	**SQL error 3624**	Database corruption	The error occurs when the system assertion checks fails due to database corruption. It can also appear due to an application bug.	High — should be addressed promptly if results due to database corruption.

#	Error Code	Error Type	Description	Severity Level
8.	**SQL error 5172**	Database-level problem	Error 5172 indicates MDF file header data corruption, and it appears when SQL is unable to attach the database due to mismatched header information.	High — requires immediate backup restore or database repair.
9.	**SQL error 8930**	Database-level issue	SQL Server encounters error 8930 when DBCC CHECKDB terminates due to metadata corruption.	High — the error cannot be repaired and requires database restoration from backup.
10.	**SQL error 8946**	Page-level problem	The error 8946 appears typically when the disk subsystem overwrites the data on a file, leading to page corruption.	The severity level depends upon the specific page and extent of corruption.
11.	**SQL error 9004**	Transaction log issue	Error 9004 is encountered due to damage in the transaction log. It appears while processing the transaction log during database rollback, recovery, or replication.	Moderate to High — can be fixed with backup restore, if available. Else, it will need transaction log repair and rebuilding.

SQL SERVER OUTAGE — DATABASE CORRUPTION SCENARIOS

This section outlines scenarios that entail business disruption in the SQL Server environment due to database-related issues. Notably, there is no or limited provision in SQL to handle these disruptive events automatically. The purpose is to shed light on such challenging situations that SQL administrators might encounter at least once, as follows:

1. DATA DELETION OR CORRUPTION WITHOUT BACKUP

This situation can arrive due to human error, software bug, or malware attack. For example, unintended use of statements like DELETE, TRUNCATE, or DROP can delete database records is considered a human error. Similarly, a malware attack can result in SQL database corruption like happened with a large multinational corporation due to NotPetya ransomware. The company's Management Information System failed to connect with the backend database, which was offline due to corruption. Read this case study to learn more. Another situation is when a software bug or database damage results in system assertion check failure, encountered as SQL error code 3624 described earlier in the table.

These issues can cause a prolonged outage if there is no backup. The administrators will need to troubleshoot the root cause with a focus on restoring the original database consistency & completeness.

2. HARDWARE FAILURE

Earlier, we read hardware failure as a major cause of SQL database corruption. The situation can lead to an outage in the absence of High Availability architecture that ensures up to 99.995% availability. Nevertheless, hardware crashes do happen, often without warning or indication, threatening the database integrity. The common hardware failure scenarios in SQL Server include disk problem, power outage, power surge, system crash due to motherboard issues, etc. The Administrator needs to fix the hardware issue after diagnosis and resolve the database inconsistencies to restore access. Hardware failure has severe implications for a SQL Server environment, particularly if there is no automatic recovery mechanism like high availability or backups.

3. DATACENTER OUTAGE

A datacenter-level outage is uncommon, but it may happen if there is a natural disaster like an earthquake, & no offsite automated failover mechanism or geo-replication is available to support disaster recovery. Notably, a datacenter outage can materialize due to database corruption, among other situations like server or network level issues. For example, a SQL database may turn into Suspect Mode while recovering from a power outage, resulting in an outage if there is no backup and recovery mechanism. Read this case study for real-world insight into the datacenter outage due to SQL database corruption.

TROUBLESHOOTING SQL DATABASE CORRUPTION

Troubleshooting a corrupted SQL Server database requires diligent planning & execution to get optimal results. Aside from the outcome, the database recovery duration is essential to meet the Recovery Time Objective (RTO). Be aware that troubleshooting SQL database issues takes time and poses specific challenges based on the root cause and extent of corruption. Therefore, following a systematic repair process is crucial to restoring the database to its original state in the shortest duration. This section provides hands-on guidance for fixing SQL database corruption through the following steps:

STEP 1: CHECK THE DATABASE INTEGRITY AND DETECT ERRORS

This step focuses on diagnosing the nature & extent of corruption by checking the SQL Server database integrity and consistency using the Database Console Commands or DBCC CHECKDB commands. The commands can check the integrity of the complete database, individual table and view, or catalog.

The general syntax of DBCC CHECKDB is given below:

```
DBCC CHECKDB
    [ ( database_name | database_id | 0
        [ , NOINDEX
        | , { REPAIR_ALLOW_DATA_LOSS | REPAIR_FAST | REPAIR_REBUILD } ]
    ) ]
    [ WITH
        {
            [ ALL_ERRORMSGS ]
            [ , EXTENDED_LOGICAL_CHECKS ]
            [ , NO_INFOMSGS ]
            [ , TABLOCK ]
            [ , ESTIMATEONLY ]
            [ , { PHYSICAL_ONLY | DATA_PURITY } ]
            [ , MAXDOP  = number_of_processors ]
        }
    ]
]
```

The DBCC CHECKDB command can take several arguments, including database_name, database_id, NOINDEX, REPAIR_ALLOW_DATA _LOSS, REPAIR_FAST, REPAIR_REBUILD, etc. The following is a short description of these arguments:

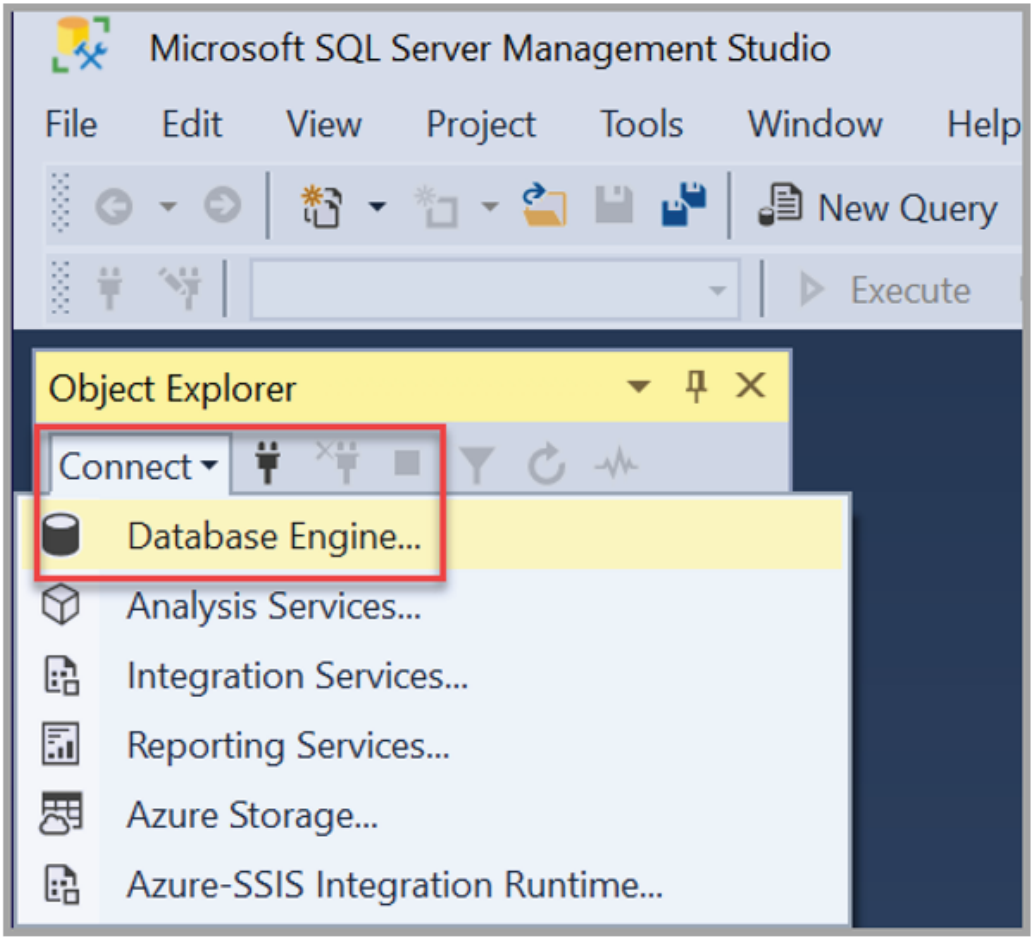
DBCC CHECKDB Argument	Description
database_name | database_id | 0	This field identifies and intakes the SQL Server database object by name or unique ID. It considers the current database if '0' is input as the value.
NOINDEX	NOINDEX parameter in the CHECKDB command specifies that non-clustered indexes in the user tables should not be intensively checked. NOINDEX reduces the execution time for the command.
REPAIR_FAST	Presence of the REPAIR_FAST parameter is for maintaining the syntax for backward compatibility. It does not perform any repair action on the database.

DBCC CHECKDB Argument	Description
REPAIR_REBUILD	This argument instructs the CHECKDB command to do quick repair tasks, such as repairing missing rows and rebuilding an index. REPAIR_REBUILD doesn't pose any risk of data loss, but it also cannot repair errors involving FILESTREAM data.
REPAIR_ALLOW_DATA_LOSS	This option specifies the command to repair all the reported errors; however, it might result in data loss. Microsoft places a caution on using REPAIR_ALLOW_DATA_LOSS as the "last resort" option as it may result in more data loss than restoring the database from the last working backup.
ALL_ERRORMSGS	This argument displays all the errors reported for the individual objects in the SQL database.
EXTENDED_LOGICAL_CHECKS	This argument extends the logical checks to the indexed view, XML indexes, and spatial indexes as applicable.
NO_INFOMSGS	This argument suppresses all the information messages.
TABLOCK	TABLOCK argument specifies the command to apply a temporary lock on the database instead of obtaining its snapshot. The arguments limits the number of checks, allowing DBCC CHECKDB to run faster.
ESTIMATEONLY	This argument simply displays the approximate tempdb space required to run the DBCC CHECKDB command.
PHYSICAL_ONLY	PHYSICAL_ONLY argument limits the command to check the integrity of only the physical structure of the page, record headers, and allocation consistency. The argument can also detect torn pages, checksum issues, and hardware malfunction.
DATA_PURITY	This argument specifies the command to check the invalid column values in the database. For example, DATA_PURITY checks the out-of-range date and time values for the datetime data type.

How to run DBCC CHECKDB command on a SQL instance?

Follow these steps to run DBCC CHECKDB using SQL Server Management Studio (SSMS):

1. Start SQL Server Management Studio and select Object Explorer > Connect > Database Engine.



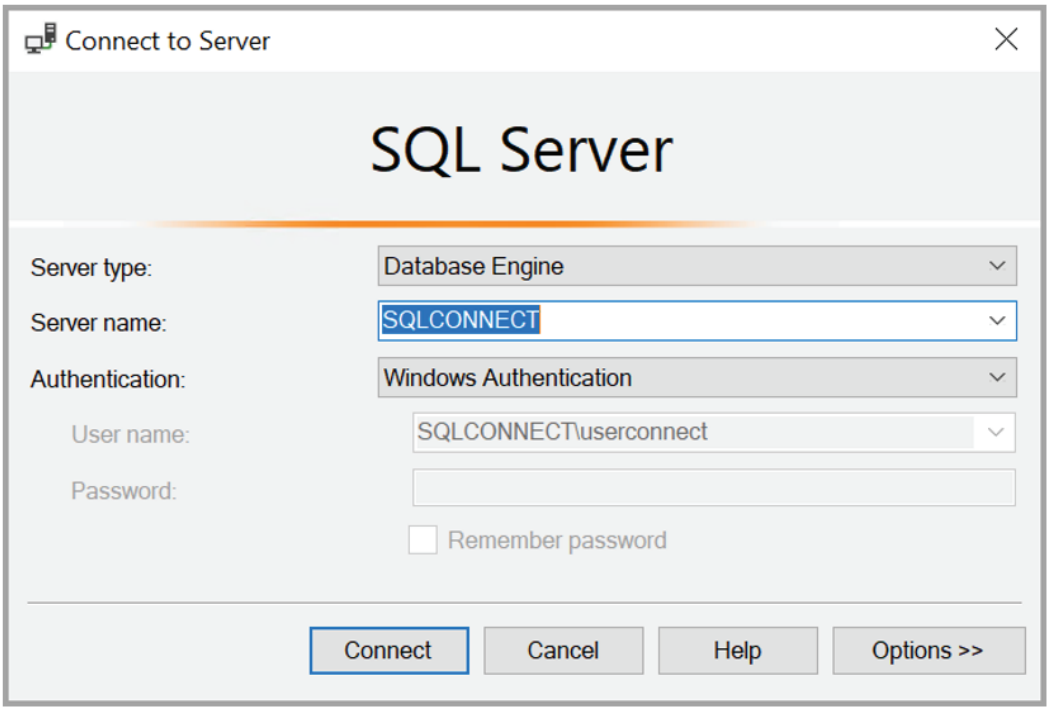
2. Enter the following information in the Connect to Server dialog box:

   a. **Server type:** select Database Engine which is the default option.

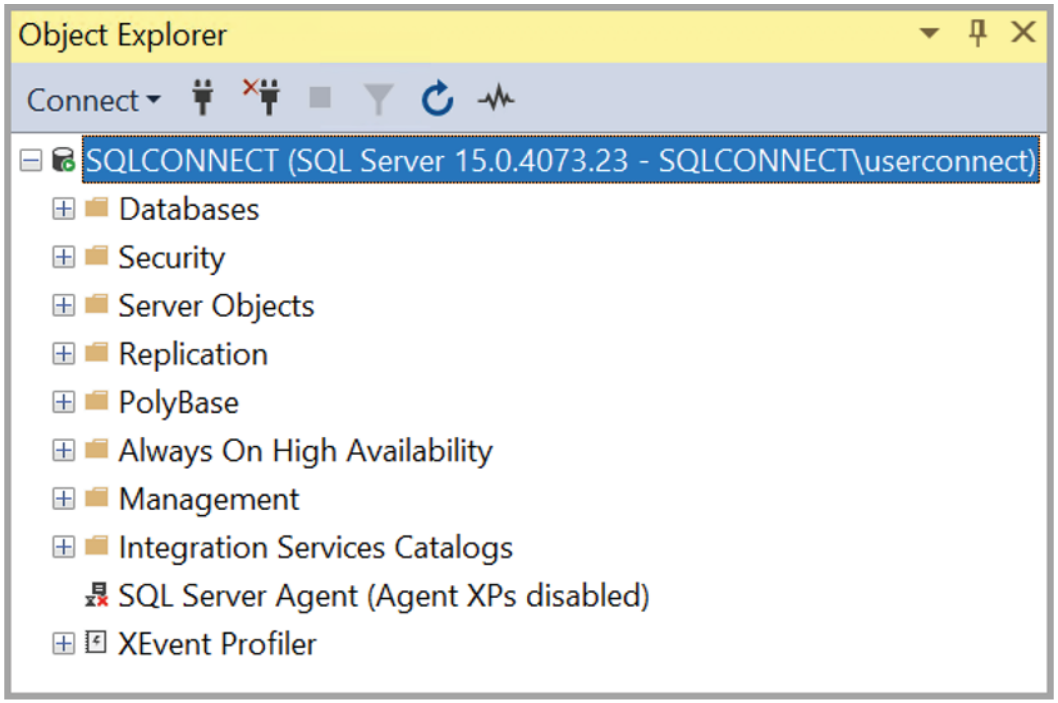
   b. **Server name:** enter the fully qualified server name.

   c. **Authentication:** use Windows Authentication as the default option.

   d. **Login ID and Password:** Enter the user ID and password to log in to the server.



3. Click Connect to establish the SQL Server connection. You will notice the server listed in Object Explorer.



4. Next, expand the "Databases" folder, and right click the database you need to query, and select New Query.

5. Run the DBCC CHECKDB command on the database, as follows:

Example 1: Use of DBCC CHECKDB to detect logical and physical errors in a database.

```
DBCC CHECKDB ('test_database')
```

Example 2: Use of WITH ALL_ERRORMSGS and NO_INFOMSGS parameters to display all the error messages while suppressing the informational messages.

```
DBCC CHECKDB (N 'test_database') WITH ALL_ERRORMSGS, NO_INFOMSGS; |
```

STEP 2: DETERMINE THE ROOT CAUSE OF CORRUPTION

After checking the database integrity and corruption errors, the next step is to determine the root cause(s), which helps avoid repeat instances and facilitates a targeted resolution if required. The below table maps the SQL database engine error codes to their root causes.

#	Error Code	Root Cause
1.	SQL error 823	Hardware problem, typically in the storage system or a driver
2.	SQL error 824	Disk failure, disk firmware issues, faulty driver, etc.
3.	SQL error 825	Severe issues in the disk hardware
4.	SQL error 605	• Failure of disk drive or other hardware • Broken page chain • Index Allocation Map (IAM) corruption
5.	SQL error 926 or 963	• Hardware issue • Input Output error • Torn page
6.	SQL error 3624	• Application bug • Data corruption
7.	SQL error 5172	• Hardware malfunctioning • Data corruption
8.	SQL error 8930	Inconsistent metadata
9.	SQL error 8946	Likely a storage disk problem if the error is encountered frequently.
10.	SQL error 9004	Might occur due to hardware, file system, and Input/output issues.

STEP 3: ADDRESS THE ROOT CAUSES OF CORRUPTION

Microsoft SQL Server documentation makes several suggestions to address the root causes of different errors. These suggestions majorly include the following user actions:

1. Review the suspect_pages in the database
2. Examine the Windows Event logs for the error messages
3. Check with the hardware vendor or OEM for compatibility issues. Use the manufacturer-supplied diagnostic utility to assess the I/O system
4. Examine the presence of **filter drivers** and evaluate whether these require updating or need to be removed altogether
5. Turn ON the PAGE_VERIFY CHECKSUM option to check page consistency
6. Use the **SQLIOSim** utility to simulate SQL Server functioning on a disk subsystem
7. For error 605, determine the tables associated with the allocation units using the following query, & then run DBCC CHECKDB for database repair (the detailed procedure is described in the following section).

```
USE`test_database`;

GO

SELECT au.allocation_unit_id, OBJECT_NAME(p.object_id) AS table_name, fg.name AS
filegroup_name,

au.type_desc AS allocation_type, au.data_pages, partition_number

FROM sys.allocation_units AS au

JOIN sys.partitions AS p ON au.container_id = p.partition_id

JOIN sys.filegroups AS fg ON fg.data_space_id = au.data_space_id

WHERE au.allocation_unit_id = `allocation_unit_id` OR au.allocation_unit_id =
`allocation_unit_id`

ORDER BY au.allocation_unit_id;

GO
```

8. Restore the database from a clean backup, if available. However, consider the backup recency and completeness against the predefined recovery SLAs.
9. Repair and fix SQL database issues using DBCC CHECKDB.

REPAIRING SQL DATABASE CORRUPTION

This section describes the use of DBCC CHECKDB commands to repair corrupted SQL Server databases. There are multiple types of arguments or parameters, including REPAIR_FAST, REPAIR_REBUILD, and REPAIR_ALLOW_DATA_LOSS to instruct a repair action using the DBCC command.

As noted earlier in this Guide, **REPAIR_FAST** does not perform an actual repair but simply preserves the syntax for backward compatibility. **REPAIR_REBUILD** is used for fixing minor issues like missing rows and is generally safe, i.e., there is no data loss. However, it cannot repair corruption errors having FILESTREAM data. Lastly, there is **REPAIR_ALLOW_DATA_LOSS** option that can resolve all types of corruption issues but will almost always result in data loss.
The following steps illustrate the use of DBCC CHECKDB with REPAIR_REBUILD and REPAIR_ALLOW_-DATA_LOSS arguments:

Step 1: Firstly, take a physical backup of the corrupted database, including the primary and secondary database files, transaction logs, memory-optimized data, etc. This step is crucial to ensure the original database remains intact even if something goes unexpected.

Step 2: Set the database to Emergency mode and then switch it over to Single-User mode using the following command—

```
ALTER DATABASE test_database SET EMERGENCY;
GO
ALTER DATABASE test_database SET SINGLE_USER WITH ROLLBACK IMMEDIATE;
GO
```

The ROLLBACK argument allows rollback of incomplete transactions.

Step 3: Run DBCC CHECKDB with REPAIR_REBUILD argument as follows:

```
DBCC CHECKDB (N'test_database',REPAIR_REBUILD) WITH NO_INFOMSGS, ALL_ERRORMSGS;
GO
ALTER DATABASE test_database SET MULTI_USER;
```

In the above example, the command will fix errors in the non-clustered indexes of the database and return the access to all the users.

Step 4: Execute DBCC CHECKDB with REPAIR_ALLOW_DATA_LOSS argument as follows:

```
DBCC CHECKDB('test_database', REPAIR_ALLOW_DATA_LOSS)
GO
ALTER DATABASE test_database SET MULTI_USER;
GO
```

The result of the command will appear like the below snippet.

```
DBCC results for 'test_database'.
Repair: The page (1:166) has been deallocated from object ID 2121058256, index ID
0, partition ID 72057594039042048, alloc unit ID 72057594043301888 (type In-row
data).
Msg 8928, Level 16, State 1, Line 1
Object ID 2121058256, index ID 0, partition ID 72057594039042048, alloc unit ID
72057594043301888 (type In-row data): Page (1:166) could not be processed.  See
other errors for details.
        The error has been repaired.
Msg 8939, Level 16, State 98, Line 1
Table error: Object ID 2121058256, index ID 0, partition ID 72057594039042048,
alloc unit ID 72057594043301888 (type In-row data), page (1:166). Test (IS_OFF
(BUF_IOERR, pBUF->bstat)) failed. Values are 29362185 and -4.
        The error has been repaired.
There are 930 rows in 14 pages for object "test_database".
        The error has been repaired.
Msg 8939, Level 16, State 98, Line 1
Table error: Object ID 2121058256, index ID 0, partition ID 72057594039042048,
alloc unit ID 72057594043301888 (type In-row data), page (1:166). Test (IS_OFF
(BUF_IOERR, pBUF->bstat)) failed. Values are 29362185 and -4.
        The error has been repaired.
There are 930 rows in 14 pages for object "test_database".
```

It is obvious that the command has eliminated the database errors. However, it also deallocated multiple pages from the Table object, which are lost forever.

SQL DATABASE REPAIR TOOL

Troubleshooting SQL database corruption is challenging and strenuous due to the vast scenarios that culminate in unforeseen errors and circumstances. A few database errors like boot page corruption and non-transient errors (like event ID 926) are considered irreparable, even using DBCC CHECKDB, and need backup restoration. However, restoring a backup database has its challenges, such as the availability of a clean backup and whether it meets the recovery SLAs, i.e., recovery point objective (RPO) and recovery time objective (RTO). Native database repair procedures using DBCC CHECKDB with REPAIR_ALLOW_DATA_LOSS can fix a majority of errors but forces the Administrator to accommodate trade-offs like permanent data loss. In this context, third-party SQL database repair tools have considerable relevance as they are purpose-built to address all types of database corruption issues with a continually evolving scope. For example, Stellar Toolkit for MS SQL is a professional software package known for repairing corrupted database file, transaction log file, & backup, enabling recovery in vast scenarios.

CLOSING NOTES

This Guide shared insights into SQL database corruption, types and scenarios, troubleshooting through database integrity checks, mapping the specific causes, and user actions. It outlined SQL database repair procedure using DBCC CHECKDB with specific arguments. Our goal is to keep this knowledge resource up-to-date and evergreen to serve as a helpful and expansive reference on SQL database corruption.

REFERENCES

SQL Database Engine Errors

DBCC CHECKDB (Transact-SQL)

Query a SQL Server using SSMS

Stellar Toolkit for MS SQL

DOWNLOAD FROM:
https://www.stellarinfo.com/sql-database-toolkit.php

+1-877-778-6087 (Tollfree)

support@stellarinfo.com

www.stellarinfo.com