https://www.sqldbachamps.com
https://github.com/PMSQLDBA/PraveenMadupu
Telegram Group→SQLDBACloud-DevOps https://t.me/+r2OYsT2qRZJkYWVl

Praveen Madupu
Mb: +91 98661 30093
Database Administrator

DBCC CHECKDB in SQL Server:

what it does, what it actually checks, Realtime/production pitfalls, best practices, pros & cons, and remediation steps you can follow when corruption shows up.

What is DBCC CHECKDB?

DBCC CHECKDB ('YourDatabase') is the built-in SQL Server integrity checker. It runs a set of checks (internally it runs CHECKALLOC, CHECKTABLE and CHECKCATALOG) to validate:

- Page and allocation consistency (allocation bitmaps, IAM pages, etc.)
- Table/index structure and B-tree consistency
- Logical catalog consistency (system tables, metadata relationships)
- (Optionally) data purity / column-level consistency depending on version and options

If corruption exists, CHECKDB reports errors and suggests repair options (for example, REPAIR_REBUILD or REPAIR_ALLOW_DATA_LOSS) — **but** repairs that allow data loss are last-resort and dangerous. The recommended path almost always is restore from verified backups.

What CHECKDB actually does (internals, at a high level)

- Validates allocation structures (pages, extents, bitmaps).
- Walks every table and index, checking physical page links and logical consistency.
- Validates system catalog consistency.
- By default, it runs *against a database snapshot* so it sees a transactionally consistent view of the database without blocking user activity.
- WITH PHYSICAL_ONLY runs a reduced set of checks (mainly basic page/format and checksum checks) and is much faster but less thorough.

Key options and what they mean

- WITH NO_INFOMSGS — suppresses non-error messages.
- WITH ALL_ERRORMSGS — prints full error messages for each problem.
- WITH PHYSICAL_ONLY — fast, minimal checks (good for daily quick checks). Does **not** check allocation/catalog thoroughly.
- WITH TABLOCK — forces CHECKDB to run directly against the live database using schema locks rather than creating a snapshot. This avoids the need to create a database snapshot but **will take locks** and can block/ be blocked.
- REPAIR_REBUILD, REPAIR_ALLOW_DATA_LOSS — options you can pass to DBCC CHECKDB(..., REPAIR_ALLOW_DATA_LOSS) but they must be used with the database in single_user mode; REPAIR_ALLOW_DATA_LOSS can remove corrupted data and therefore can *lose* data — prefer restores.

Realtime / production issues you must know about

1. **Snapshot creation can fail**
    - By default CHECKDB creates a database snapshot to run against. The snapshot is *sparse copy-on-write* and requires disk space for changed pages. If the snapshot can't be created (insufficient disk space, snapshot file location issues, VSS interactions), CHECKDB will fail.
    - Mitigation: ensure there is enough free space on the volume where the database files reside (or configured snapshot location), or run with TABLOCK if appropriate (but see next point).
2. **Resource consumption (I/O, CPU, tempdb / snapshot growth)**
    - CHECKDB is I/O and CPU intensive and can generate lots of read/write and temp file activity. On large databases it can run for hours.
    - The snapshot's copy-on-write behavior increases IO for active workloads (writes cause extra IO to populate snapshot).
    - Mitigation: run during low activity windows or against a restored copy; use PHYSICAL_ONLY for daily quick checks.
3. **Blocking and locking**
    - Default snapshot approach avoids blocking. But running with TABLOCK avoids snapshot creation but places schema locks that will block user transactions (may be unacceptable).
    - Long running TABLOCK can halt application activity. Avoid unless you understand the impact.
4. **tempdb pressure**
    - CHECKDB and the snapshot mechanism may create tempdb pressure (especially on versions doing additional checks). Ensure tempdb has capacity and good IO.
5. **Restore vs Repair tradeoffs**
    - REPAIR_ALLOW_DATA_LOSS can appear attractive but may corrupt relationships or silently drop rows/indexes. In production, often preferable to restore from a clean backup than to run REPAIR_ALLOW_DATA_LOSS.
6. **False sense of safety**
    - Passing PHYSICAL_ONLY daily but never running full checks can miss logical or allocation corruptions. Balance frequency vs depth.

Best practices (practical checklist)

1. **Enable page checksums**

    ALTER DATABASE <db> SET PAGE_VERIFY CHECKSUM;
    - This catches many I/O/transmission errors early and ensures that the storage subsystem corruption is detected when the page is read.
2. **Run regular DBCC CHECKDB**
    - Typical pattern:
        - **Daily**: DBCC CHECKDB WITH PHYSICAL_ONLY (fast smoke-check)
        - **Weekly (or nightly for critical DBs)**: full DBCC CHECKDB (no PHYSICAL_ONLY)

- Frequency depends on criticality: mission-critical → more frequent.

3. **Run CHECKDB against a restored copy (preferred for very large DBs)**
   - Offload heavy integrity checks by restoring the latest full backup (and necessary logs) to a separate server and run CHECKDB there. That avoids production performance impact.
4. **Ensure sufficient disk for snapshots**
   - Snapshot (sparse file) may need space equal to changed pages during the run. Always provide comfortable free space on volumes that house the DB files.
5. **Use TABLOCK only when you understand locking**
   - TABLOCK can be used when snapshot creation fails, but expect blocking. Plan maintenance windows.
6. **Automate and alert**
   - Automate checks via maintenance jobs (SQL Agent or external orchestration) and alert on non-zero errors. Capture output (e.g., redirect DBCC output to table/file and parse for "repair" suggestions or error codes).
7. **Have a tested restore plan**
   - The golden rule: if corruption occurs, restore from a known-good backup. Regularly test restores onto a separate server. Don't rely on repairs as a primary recovery method.
8. **Use trusted maintenance scripts**
   - Use community-tested maintenance scripts (e.g., Ola Hallengren's maintenance solution is common) or well-tested in-house scripts. They handle CHECKDB scheduling, logging, and error handling.
9. **Monitor health proactively**
   - Monitor SQL Server error logs, Windows application/system logs, and hardware alerts. Run consistency checks after storage maintenance, power events, or hardware failures.
10. **Protect your backups**

- Corruption can get propagated to backups if backups are taken after corruption occurs. Keep multiple backup generations and verify backups (e.g., RESTORE VERIFYONLY, regular test restores).

Typical maintenance schedule example

- Daily (off-peak): DBCC CHECKDB ('DB') WITH PHYSICAL_ONLY, NO_INFOMSGS.
- Weekly (full maintenance window): DBCC CHECKDB ('DB') WITH ALL_ERRORMSGS, NO_INFOMSGS for all databases.
- Monthly: full restore test of a critical database to a separate server and run CHECKDB there.

Advantages of using DBCC CHECKDB

- **Detects corruption early** — helps you know about allocation and page corruption before it's user-visible.
- **Comprehensive** — checks allocation, index and data integrity, and catalog consistency.
- **Built into SQL Server** — no third-party tool needed.
- **Provides repair guidance** — indicates repair options and severity.
- **Snapshot-based** — usually nonblocking to production workloads (if snapshot can be created).

Disadvantages / limitations

- **Resource-intensive** — can be costly on I/O/CPU for large DBs.
- **Snapshot failure risk** — requires disk capacity; if snapshot fails you must choose TABLOCK or other approaches.
- **PHYSICAL_ONLY is limited** — if you only rely on PHYSICAL_ONLY you may miss logical/cross-structure corruptions.

- **Repair options can cause data loss** — REPAIR_ALLOW_DATA_LOSS can and will remove problem data.
- **Doesn't fix the root cause** — it only reports corruption; root causes (storage, firmware, network, drivers) must be diagnosed and corrected.

When CHECKDB finds corruption — recommended steps

1. **Capture and secure the CHECKDB output**. Save the full output (ALL_ERRORMSGS).
2. **Quarantine the server if needed**. If corruption is widespread, consider taking the DB offline to prevent further damage.
3. **Do NOT immediately run REPAIR_ALLOW_DATA_LOSS as a first option**. Instead:
    - Identify affected objects and assess business impact.
    - Try restoring from the most recent clean backup: restore to a separate instance and run CHECKDB there to confirm backup integrity.
    - If a clean backup exists close to the corruption time, restore to production.
4. **If no good backup exists**: consider REPAIR_REBUILD (non-data losing) first if suggested and applicable. REPAIR_ALLOW_DATA_LOSS is last resort — take full backups before running it and be prepared for data loss.
5. **Engage Storage / Hardware teams** to identify the root cause (controller, firmware, RAID issues, bad NICs, SAN path errors, disk errors).
6. **After recovery**, enable checks, run full CHECKDB, and monitor closely.

Sample T-SQL commands and examples

Quick smoke check:

```
-- fast, minimal checks:
    DBCC CHECKDB ('MyDatabase') WITH PHYSICAL_ONLY, NO_INFOMSGS;
```

Full check with verbose error messages:

```
    DBCC CHECKDB ('MyDatabase') WITH ALL_ERRORMSGS, NO_INFOMSGS;
    GO
```

If snapshot creation fails and you accept locking repercussions:

```
DBCC CHECKDB ('MyDatabase') WITH TABLOCK, ALL_ERRORMSGS;
```

Capture results into a table (example pattern):

```
    CREATE TABLE dbo.CheckDBLog (
     RunDate datetime2,
     DatabaseName sysname,
     OutputText nvarchar(max)
    );
-- example of executing and inserting output would require xp_readerrorlog or capturing via SQL Agent job step output.
```

(Use SQL Agent job step output capture or PowerShell to capture DBCC output reliably.)

Additional practical tips

- **Keep SQL Server and storage firmware/drivers patched** — many corruptions arise from underlying hardware or driver bugs.
- **Use RAID/Storage that provides redundancy and checks** (and periodically scrub disks if supported).

- **Avoid instant file initialization for data files only**; it helps performance but it doesn't affect checks — just be aware of its effects.
- **Check windows and SQL Server event logs** around times corruption appears — often storage subsystem or OS warnings precede it.
- **Consider separate maintenance servers** for restored-checks for very large or critical databases.

Summary:

- DBCC CHECKDB is the canonical SQL Server database integrity checker — run it regularly.
- Use PHYSICAL_ONLY daily for quick checks and full CHECKDB periodically.
- Prefer running full CHECKDB on a restored copy to avoid production load or blocking.
- Ensure page checksums and a good backup & restore testing strategy are in place.
- If corruption appears, prefer restore from good backup; REPAIR_ALLOW_DATA_LOSS is last resort and risky.
- Monitor disk space (snapshot usage), tempdb, and overall server resources to avoid CHECKDB failures.

Maintenance job script (SQL Agent job step + T-SQL) to tune environment (DB sizes/business windows)

Complete SQL Server Maintenance Job script for **DBCC CHECKDB**, for 3 database size categories:

DB Size	CHECKDB Frequency	CHECK Type	Target Window
100 GB	**Weekly full**, **Daily PHYSICAL_ONLY**	Full weekly + Fast daily	Overnight window
500 GB	**Weekly full**, **Daily PHYSICAL_ONLY**	Full weekly (off-peak), fast daily	Maintenance window required
1 TB	**Full CHECKDB on restored copy**, **Daily PHYSICAL_ONLY on prod**	Because full CHECKDB is heavy	Must schedule on DR / restore server

✅ **Job Logic you will get (multi-step SQL Agent Job)**

This job will:

1. **Run PHYSICAL_ONLY every day** at night (fast, minimal IO)
2. **Run FULL CHECKDB weekly on production** for DBs ≤ 500 GB
3. **Run FULL CHECKDB only on DR copy** for DBs ≥ 1 TB
4. **Log results** into DBA_CheckDB_Log table for auditing
5. **Email Alerts automatically** on corruption findings

✅ **Create a Logging Table (Run once on msdb or DBA database)**

```
USE DBA;
GO
CREATE TABLE dbo.DBA_CheckDB_Log (
   LogID INT IDENTITY PRIMARY KEY,
   DatabaseName SYSNAME,
   CheckType VARCHAR(30),
   RunDate DATETIME DEFAULT GETDATE(),
   ResultText NVARCHAR(MAX)
);
```

✅ **Step 1 — SQL Agent Job (Daily) — PHYSICAL_ONLY for all DBs**

Job Name: DBCC CHECKDB Daily (PHYSICAL_ONLY)

Job Step T-SQL:

```
DECLARE @DB NVARCHAR(255), @SQL NVARCHAR(MAX);
DECLARE db_cursor CURSOR FOR
SELECT name FROM sys.databases
WHERE state = 0 AND database_id > 4;   -- Excluding master, model, msdb, tempdb

OPEN db_cursor;
FETCH NEXT FROM db_cursor INTO @DB;

WHILE @@FETCH_STATUS = 0
BEGIN
   SET @SQL = '
      DECLARE @Result NVARCHAR(MAX);
      BEGIN TRY
         DBCC CHECKDB ([' + @DB + ']) WITH PHYSICAL_ONLY, NO_INFOMSGS;
         SET @Result = ''OK'';
      END TRY
      BEGIN CATCH
         SET @Result = ERROR_MESSAGE();
      END CATCH;
      INSERT INTO DBA.dbo.DBA_CheckDB_Log(DatabaseName, CheckType, ResultText)
      VALUES (''' + @DB + ''', ''PHYSICAL_ONLY'', @Result);
   ';
   EXEC (@SQL);
```

```
    FETCH NEXT FROM db_cursor INTO @DB;
END

CLOSE db_cursor;
DEALLOCATE db_cursor;
```

📌 **Schedule:** Daily at **2:00 AM**

✅ **Step 2 — Weekly FULL CHECKDB (Only for DBs <= 500 GB)**

Job Name: DBCC CHECKDB Weekly Full

Job Step T-SQL:

```
DECLARE @DB NVARCHAR(255), @SQL NVARCHAR(MAX);
;WITH DBList AS
(
   SELECT name, sizeGB = (SUM(size)*8)/1024/1024
   FROM sys.master_files
   GROUP BY name
)
SELECT name INTO #DBToCheck FROM DBList WHERE sizeGB <= 500;  -- skip 1TB DBs
DECLARE db_cursor CURSOR FOR SELECT name FROM #DBToCheck;
OPEN db_cursor;
FETCH NEXT FROM db_cursor INTO @DB;
WHILE @@FETCH_STATUS = 0
BEGIN
   SET @SQL = '
      DECLARE @Result NVARCHAR(MAX);
      BEGIN TRY
         DBCC CHECKDB ([' + @DB + ']) WITH NO_INFOMSGS, ALL_ERRORMSGS;
         SET @Result = ''OK'';
      END TRY
      BEGIN CATCH
         SET @Result = ERROR_MESSAGE();
      END CATCH;

      INSERT INTO DBA.dbo.DBA_CheckDB_Log(DatabaseName, CheckType, ResultText)
      VALUES (''' + @DB + ''', ''FULL_CHECK'', @Result);
   ';
   EXEC (@SQL);

   FETCH NEXT FROM db_cursor INTO @DB;
END

CLOSE db_cursor;
DEALLOCATE db_cursor;
DROP TABLE #DBToCheck;
```

📌 **Schedule:** Sunday 2:00 AM (low-activity window)

✅ **Step 3 — 1 TB DB CHECKDB Strategy (Important)**

For the **1 TB database**, do this instead:

Location	CHECKDB Type
Prod	Daily PHYSICAL_ONLY (already covered)
DR Server / Restore Server	Weekly FULL CHECKDB on restored copy

That offloads heavy IO from production and avoids snapshot issues.

✅ Optional — Corruption Email Alert (Highly Recommended)

Create an alert job step:

```
IF EXISTS (SELECT 1 FROM DBA.dbo.DBA_CheckDB_Log
      WHERE RunDate >= DATEADD(DAY,-1,GETDATE())
      AND ResultText <> 'OK')
BEGIN
   EXEC msdb.dbo.sp_send_dbmail
     @profile_name = 'DBA_MAIL',
     @recipients = 'dba-team@yourcompany.com',
     @subject = 'URGENT: SQL Corruption Detected',
     @body = 'DBCC CHECKDB found corruption. Check DBA_CheckDB_Log immediately.';
END
```

✅ Why this strategy is the best fit

DB Size	Why This Plan Works
100 GB	Full CHECKDB is fast, snapshot manageable
500 GB	Weekly CHECKDB OK, but avoid daily full
1 TB	Full CHECKDB on prod is too expensive — offload to DR

Complete SQL Agent maintenance package that uses **SQL Agent Operator notifications.**

This includes:

- Logging table (one-time create)
- Operator creation (SQL Agent operator)
- Three jobs:
    1. **Daily PHYSICAL_ONLY** — runs on *all* prod DBs (fast smoke check)
    2. **Weekly FULL** — runs full CHECKDB for DBs **≤ 500 GB** (prod)
    3. **DR Full CHECKDB** — intended to run on your DR/restore server for the **1 TB** DB (restore latest backup then run full CHECKDB)
- Schedules for each job
- Job notification configuration to alert the operator on **job failure**
- A cleanup / rollback script

Before you run this

1. Make sure **Database Mail** is configured on the instance and SQL Agent is enabled to use Database Mail (SQL Agent Properties → Alert System → enable and choose DB Mail profile).
2. Modify variables like @OperatorEmail, @DBA_DB (DB name for logging), and DR restore paths to suit your environment.
3. Test on a non-production instance first.

1) Create logging DB/table (run once)

You can store logs in an existing DBA database; change DBA below if needed.

```
-- Create DBA database (optional) and logging table (run once)
IF DB_ID('DBA') IS NULL
BEGIN
   CREATE DATABASE DBA;
   -- Optionally configure file locations, growth, etc.
END;
GO

USE DBA;
GO

IF OBJECT_ID('dbo.DBA_CheckDB_Log','U') IS NOT NULL
   DROP TABLE dbo.DBA_CheckDB_Log;
GO

CREATE TABLE dbo.DBA_CheckDB_Log (
   LogID INT IDENTITY PRIMARY KEY,
   DatabaseName SYSNAME NOT NULL,
   CheckType VARCHAR(30) NOT NULL,
   RunDate DATETIME2 DEFAULT SYSUTCDATETIME(),
   ResultText NVARCHAR(MAX) NULL
);
GO
```

2) Create SQL Agent Operator

Change @OperatorName / @OperatorEmail to your operator values.

```
USE msdb;
GO

DECLARE @OperatorName SYSNAME = N'DBA_Operator';
DECLARE @OperatorEmail NVARCHAR(256) = N'dba-team@yourcompany.com';

-- If operator exists, drop (optional) and recreate
IF EXISTS (SELECT 1 FROM msdb.dbo.sysoperators WHERE name = @OperatorName)
BEGIN
   PRINT 'Operator exists - skipping create';
END
ELSE
BEGIN
   EXEC msdb.dbo.sp_add_operator
      @name = @OperatorName,
      @enabled = 1,
      @email_address = @OperatorEmail;
   PRINT 'Operator created';
END
GO
```

Make sure the operator receives emails (Database Mail working + SQL Agent Alert System enabled).

3) Utility: helper to compute DB size (GB) — used inside jobs

(This is used within job steps; included here for reference — no execution required.)

```
-- Example query to check database sizes in GB:
SELECT
   name,
   sizeGB = SUM(size) * 8.0 / 1024 / 1024
FROM sys.master_files
WHERE database_id > 4
GROUP BY name
ORDER BY sizeGB DESC;
```

4) Job A — Daily PHYSICAL_ONLY (fast) — create via T-SQL

This job iterates all online databases (excluding system DBs) and runs DBCC CHECKDB WITH PHYSICAL_ONLY, logs results, and notifies operator on failure.

```
USE msdb;
GO
DECLARE @JobName SYSNAME = N'DBCC_CHECKDB_Daily_PHYSICAL_ONLY';
DECLARE @Owner SYSNAME = SUSER_SNAME();  -- change owner if needed
DECLARE @OperatorName SYSNAME = N'DBA_Operator';

-- Clean up if job exists (optional)
IF EXISTS (SELECT 1 FROM msdb.dbo.sysjobs WHERE name = @JobName)
BEGIN
   EXEC msdb.dbo.sp_delete_job @job_name = @JobName;
END
```

```
-- Create job
EXEC msdb.dbo.sp_add_job
   @job_name = @JobName,
   @owner_login_name = @Owner,
   @enabled = 1,
   @description = N'Daily DBCC CHECKDB PHYSICAL_ONLY for all user DBs',
   @notify_level_email = 2,              -- 0 = none, 1 = success, 2 = failure, 3 = completion
   @notify_email_operator_name = @OperatorName;
GO

-- Add single T-SQL step that loops DBs and logs results
EXEC msdb.dbo.sp_add_jobstep
   @job_name = @JobName,
   @step_name = N'Run PHYSICAL_ONLY on user DBs',
   @subsystem = N'TSQL',
   @command = N'
DECLARE @DB SYSNAME, @sql NVARCHAR(MAX), @result NVARCHAR(MAX);
DECLARE dbcur CURSOR LOCAL FAST_FORWARD FOR
   SELECT name FROM sys.databases WHERE state = 0 AND database_id > 4; -- online, user DBs

OPEN dbcur;
FETCH NEXT FROM dbcur INTO @DB;
WHILE @@FETCH_STATUS = 0
BEGIN
   BEGIN TRY
      SET @sql = N''DBCC CHECKDB ([' + @DB + ']) WITH PHYSICAL_ONLY, NO_INFOMSGS;'';
      EXEC sp_executesql @sql;
      SET @result = N''OK'';
   END TRY
   BEGIN CATCH
      SET @result = ERROR_MESSAGE();
   END CATCH;

   INSERT INTO DBA.dbo.DBA_CheckDB_Log(DatabaseName, CheckType, ResultText)
   VALUES (@DB, N''PHYSICAL_ONLY'', @result);

   FETCH NEXT FROM dbcur INTO @DB;
END
CLOSE dbcur;
DEALLOCATE dbcur;
',
   @on_success_action = 1,  -- go to next step / quit with success (only one step)
   @on_fail_action = 2;     -- quit with failure
GO

-- Schedule: Daily at 02:00 AM
EXEC msdb.dbo.sp_add_schedule
   @schedule_name = N'Sched_Daily_2AM_PHYSICAL_ONLY',
   @freq_type = 4,          -- daily
   @freq_interval = 1,      -- every 1 day
   @active_start_time = 20000; -- 02:00:00
```

```
-- Attach schedule to job
EXEC msdb.dbo.sp_attach_schedule
   @job_name = @JobName,
   @schedule_name = N'Sched_Daily_2AM_PHYSICAL_ONLY';

-- Add job to server
EXEC msdb.dbo.sp_add_jobserver
   @job_name = @JobName,
   @server_name = @@SERVERNAME;
GO
```

5) Job B — Weekly FULL CHECKDB (prod DBs ≤ 500 GB)

This job finds DBs whose total file sizes are ≤ 500 GB and runs full DBCC CHECKDB with ALL_ERRORMSGS, logs results and notifies operator on failure. **Schedule**: weekly (Sunday 02:00 AM).

```
USE msdb;
GO
DECLARE @JobName SYSNAME = N'DBCC_CHECKDB_Weekly_Full_LE_500GB';
DECLARE @Owner SYSNAME = SUSER_SNAME();
DECLARE @OperatorName SYSNAME = N'DBA_Operator';

IF EXISTS (SELECT 1 FROM msdb.dbo.sysjobs WHERE name = @JobName)
BEGIN
   EXEC msdb.dbo.sp_delete_job @job_name = @JobName;
END

EXEC msdb.dbo.sp_add_job
   @job_name = @JobName,
   @owner_login_name = @Owner,
   @enabled = 1,
   @description = N'Weekly FULL DBCC CHECKDB for DBs with size <= 500 GB',
   @notify_level_email = 2,
   @notify_email_operator_name = @OperatorName;
GO
EXEC msdb.dbo.sp_add_jobstep
   @job_name = @JobName,
   @step_name = N'Run FULL CHECKDB on DBs <= 500GB',
   @subsystem = N''TSQL'',
   @command = N'
DECLARE @DB SYSNAME, @sql NVARCHAR(MAX), @result NVARCHAR(MAX);

-- Build list of DBs <= 500 GB
CREATE TABLE #DBList(name SYSNAME, sizeGB DECIMAL(10,2));

INSERT INTO #DBList (name,sizeGB)
SELECT name, SUM(size)*8.0/1024/1024 AS sizeGB
FROM sys.master_files
WHERE database_id > 4
GROUP BY name
HAVING SUM(size)*8.0/1024/1024 <= 500; -- Filter <= 500 GB
```

```
DECLARE cur CURSOR LOCAL FAST_FORWARD FOR SELECT name FROM #DBList;
OPEN cur;
FETCH NEXT FROM cur INTO @DB;
WHILE @@FETCH_STATUS = 0
BEGIN
   BEGIN TRY
      SET @sql = N''DBCC CHECKDB ([' + @DB + ']) WITH ALL_ERRORMSGS, NO_INFOMSGS;'';
      EXEC sp_executesql @sql;
      SET @result = N''OK'';
   END TRY
   BEGIN CATCH
      SET @result = ERROR_MESSAGE();
   END CATCH;

   INSERT INTO DBA.dbo.DBA_CheckDB_Log(DatabaseName, CheckType, ResultText)
   VALUES (@DB, N''FULL_CHECK'', @result);

   FETCH NEXT FROM cur INTO @DB;
END

CLOSE cur;
DEALLOCATE cur;
DROP TABLE #DBList;
',
   @on_success_action = 1,
   @on_fail_action = 2;
GO

-- Schedule: Weekly Sunday 02:00 AM
EXEC msdb.dbo.sp_add_schedule
   @schedule_name = N'Sched_Weekly_Sun_2AM_FULL',
   @freq_type = 8,           -- weekly
   @freq_interval = 1,       -- every week
   @freq_recurrence_factor = 1,
   @active_start_time = 20000, -- 02:00:00
   @active_start_date = 20000101;
EXEC msdb.dbo.sp_attach_schedule
   @job_name = @JobName,
   @schedule_name = N'Sched_Weekly_Sun_2AM_FULL';

EXEC msdb.dbo.sp_add_jobserver
   @job_name = @JobName,
   @server_name = @@SERVERNAME;
GO
```

6) Job C — DR Full CHECKDB for 1 TB DB (restore + CHECKDB on DR server)

Important: This job is **meant to run on your DR/restore instance** (not production). Workflow:

1. Restore latest full backup (and required log backups if needed) to a database named like DBName_CheckDB_Restore
2. Run DBCC CHECKDB FULL on the restored copy
3. Drop the restored copy (optional)
4. Log results and alert operator on failure

You must customize the backup path, backup naming convention, and retention policy. Below is a template you must adapt.

```
-- This script should be deployed on the DR/restore server.
        USE msdb;
        GO
        DECLARE @JobName SYSNAME = N'DR_Restore_And_CheckDB_Full_1TB_DB';
        DECLARE @Owner SYSNAME = SUSER_SNAME();
        DECLARE @OperatorName SYSNAME = N'DBA_Operator';
        IF EXISTS (SELECT 1 FROM msdb.dbo.sysjobs WHERE name = @JobName)
        BEGIN
           EXEC msdb.dbo.sp_delete_job @job_name = @JobName;
        END

        EXEC msdb.dbo.sp_add_job
           @job_name = @JobName,
           @owner_login_name = @Owner,
           @enabled = 1,
           @description = N'Restore latest backup of big DB and run FULL DBCC CHECKDB on restored copy',
           @notify_level_email = 2,
           @notify_email_operator_name = @OperatorName;
        GO
        EXEC msdb.dbo.sp_add_jobstep
           @job_name = @JobName,
           @step_name = N'Restore latest backup and run CHECKDB',
           @subsystem = N''TSQL'',
           @command = N'
        -- CONFIGURE THESE VARIABLES BEFORE RUNNING:
        DECLARE @SourceDB SYSNAME = N''Your1TBDB'';          -- original DB logical name
        DECLARE @RestoreDB SYSNAME = @SourceDB + N''_CheckDB''; -- restored DB name on DR
        DECLARE @BackupDir NVARCHAR(4000) = N''\\backup-share\sqlbackups\Your1TBDB\''; -- network share or local path
        DECLARE @LatestFull NVARCHAR(4000);

-- find latest full backup file based on naming convention (customize as needed)
-- This example assumes backup files like Your1TBDB_Full_20251020.bak
-- You may implement a more robust method (backup history tables, maintenance cleanup).
-- For simple cases, if backup files are on local disk on DR and you can find by XP, adapt accordingly.
-- Below is a placeholder; replace with the logic you use to pick the correct backup.

SET @LatestFull = (SELECT TOP 1 [physical_device_name]
            FROM msdb..backupmediafamily mf
            JOIN msdb..backupset bs ON mf.media_set_id = bs.media_set_id
            WHERE bs.database_name = @SourceDB AND bs.type = ''D''
            ORDER BY bs.backup_finish_date DESC);
IF @LatestFull IS NULL
BEGIN
   RAISERROR('No full backup found for database %s', 16, 1, @SourceDB);
   RETURN;
END
```

-- Restore full backup WITH REPLACE to restore DB

```
        DECLARE @restoreSQL NVARCHAR(MAX) = N''RESTORE DATABASE ['' + @RestoreDB + ''] FROM DISK = N'''' + @LatestFull + ''''
        + N'' WITH MOVE '''' + @SourceDB + '''' TO N'''' + REPLACE(@LatestFull, '.bak','_data.mdf') + '''', MOVE ''' + @SourceDB + '_log''
        TO N''' + REPLACE(@LatestFull, '.bak','_log.ldf') + ''', REPLACE, STATS=5;'';

        BEGIN TRY
          EXEC sp_executesql @restoreSQL;
        END TRY
        BEGIN CATCH
          INSERT INTO DBA.dbo.DBA_CheckDB_Log(DatabaseName, CheckType, ResultText)
          VALUES (@SourceDB, N''DR_RESTORE'', ERROR_MESSAGE());
          THROW;
        END CATCH;

        -- Run FULL DBCC CHECKDB on restored DB
        BEGIN TRY
          DBCC CHECKDB ([' + @RestoreDB + ']) WITH ALL_ERRORMSGS, NO_INFOMSGS;
          INSERT INTO DBA.dbo.DBA_CheckDB_Log(DatabaseName, CheckType, ResultText)
          VALUES (@RestoreDB, N''DR_FULL_CHECKDB'', N''OK'');
        END TRY
        BEGIN CATCH
          INSERT INTO DBA.dbo.DBA_CheckDB_Log(DatabaseName, CheckType, ResultText)
          VALUES (@RestoreDB, N''DR_FULL_CHECKDB'', ERROR_MESSAGE());
          THROW;
        END CATCH;

-- Optionally drop restored DB to save space (uncomment if desired)
-- ALTER DATABASE [' + @RestoreDB + '] SET SINGLE_USER WITH ROLLBACK IMMEDIATE;
-- DROP DATABASE [' + @RestoreDB + '];
',
  @on_success_action = 1,
  @on_fail_action = 2;
GO
-- Schedule: Weekly (choose a low-usage day/time)
EXEC msdb.dbo.sp_add_schedule
  @schedule_name = N'DR_Sched_Weekly_Full_Check',
  @freq_type = 8,          -- weekly
  @freq_interval = 1,
  @active_start_time = 30000; -- e.g., 03:00:00
EXEC msdb.dbo.sp_attach_schedule
  @job_name = @JobName,
  @schedule_name = N'DR_Sched_Weekly_Full_Check';

EXEC msdb.dbo.sp_add_jobserver
  @job_name = @JobName,
  @server_name = @@SERVERNAME;
GO
```

Note: The DR script contains placeholders for selecting the correct backup file — please replace with your method (backup share naming convention, or using msdb backup tables with accessible physical_device_name entries, or a custom file picker).

7) Optional: Alert on specific CHECKDB errors (SQL Agent Alert)

If you prefer to create an **SQL Server Alert** for DBCC error numbers indicating corruption (for example, error numbers like 824, 825, 823 etc), you can add alerts that notify the operator immediately when such errors appear in the error log.

-- Example: create alert for error 824 (I/O error / logical consistency)

```
USE msdb;
GO
EXEC msdb.dbo.sp_add_alert
   @name = N'SQL_Error_824',
   @message_id = 824,
   @severity = 0,
   @enabled = 1,
   @delay_between_responses = 0,
   @include_event_description_in = 1,
   @job_id = NULL;

-- Notify operator
EXEC msdb.dbo.sp_add_notification
   @alert_name = N'SQL_Error_824',
   @operator_name = N'DBA_Operator',
   @notification_method = 1;  -- 1 = Email
GO
```

You can add similar alerts for 823, 825, or other storage/IO related errors.

8) Cleanup / Rollback script

If you want to remove the jobs and operator:

```
USE msdb;
GO
-- Delete jobs
DECLARE @jobs TABLE (name SYSNAME);
INSERT INTO @jobs VALUES
('DBCC_CHECKDB_Daily_PHYSICAL_ONLY'),
('DBCC_CHECKDB_Daily_PHYSICAL_ONLY'),
('DBCC_CHECKDB_Weekly_Full_LE_500GB'),
('DR_Restore_And_CheckDB_Full_1TB_DB');

DECLARE @name SYSNAME;
DECLARE cur CURSOR FAST_FORWARD FOR SELECT name FROM @jobs;
OPEN cur;
FETCH NEXT FROM cur INTO @name;
WHILE @@FETCH_STATUS = 0
BEGIN
   IF EXISTS (SELECT 1 FROM msdb.dbo.sysjobs WHERE name = @name)
   BEGIN
      EXEC msdb.dbo.sp_delete_job @job_name = @name;
   END
   FETCH NEXT FROM cur INTO @name;
END
CLOSE cur;
DEALLOCATE cur;
```

```
-- Remove operator (optional)
DECLARE @OperatorName SYSNAME = N'DBA_Operator';
IF EXISTS (SELECT 1 FROM msdb.dbo.sysoperators WHERE name = @OperatorName)
BEGIN
   EXEC msdb.dbo.sp_delete_operator @name = @OperatorName;
END

-- Drop logging table (optional)
USE DBA;
IF OBJECT_ID('dbo.DBA_CheckDB_Log','U') IS NOT NULL
   DROP TABLE dbo.DBA_CheckDB_Log;
GO
```

9) GUI steps (if you prefer SQL Agent GUI)

1. Configure Database Mail and set a profile.
2. In SQL Server Agent → Right-click → Properties → Alert System → Check “Enable mail profile” and choose the DB Mail profile. Restart SQL Agent if prompted.
3. In SQL Server Agent → Operators → New Operator → set name + email.
4. Jobs → New Job → add Steps (T-SQL from above), Schedules and in Notifications tab choose “When the job fails” → notify Operator (the operator you created).
5. For DR job, create the job on the DR instance and schedule it weekly.

10) Practical notes & tuning

- For **100 GB** DBs: full weekly on prod is OK (full CHECKDB typically completes faster and snapshot creation is manageable).
- For **500 GB** DBs: weekly full on prod may be OK during maintenance window, but monitor snapshot disk usage and tempdb. Consider staggering DBs across different nights if you have many.
- For **1 TB** DBs: **prefer restore-based CHECKDB** (offload to DR). Restoring and running CHECKDB on a separate server prevents heavy I/O on production, avoids snapshot failures, and is the recommended approach for very large DBs.
- Consider staggering full CHECKDB runs across multiple nights so not all large DBs run the same night.
- If DBCC CHECKDB snapshot fails due to space, the job will fail — consider adding logic to detect snapshot capability or fallback to TABLOCK only during maintenance windows (but be careful: TABLOCK will block).
- Keep a rolling retention of logs in DBA_CheckDB_Log — implement cleanup (e.g., delete rows older than X days).

DBCC CHECKDB strategy for Small / Medium / Very Large databases using Ola Hallengren's Maintenance Solution:

✅ CHECKDB Strategy by DB Size
✅ Ola Hallengren CMD Parameters for each size
✅ SQL Agent Job Step scripts (CommandLine for Ola jobs)
✅ Real-world best practices for VLDB (1 TB+)
✅ Notes on scheduling + offloading to DR

📌 1. CHECKDB Strategy by Database Size

DB Size	Recommended CHECKDB Plan	Why
Small (≤ 100 GB)	Full CHECKDB **Weekly** + PHYSICAL_ONLY **Daily**	Low overhead, fast snapshot
Medium (100 – 500 GB)	**Weekly Full** + PHYSICAL_ONLY **Daily**	Full is OK but schedule off-peak
Very Large (500 GB – Multi TB)	**PHYSICAL_ONLY Daily on Prod** + **Full CHECKDB Weekly on Restore/DR Server**	Full CHECKDB is expensive; avoid on prod

📌 2. Ola Hallengren Job Steps (DBCC CHECKDB)

Ola's command: EXECUTE dbo.DatabaseIntegrityCheck

✅ A) Small DBs (≤ 100 GB)

Daily PHYSICAL_ONLY job:

```
EXECUTE dbo.DatabaseIntegrityCheck
@Databases = 'USER_DATABASES',
@CheckCommands = 'CHECKDB',
@PhysicalOnly = 'Y';
```

Weekly FULL CHECKDB job:

```
EXECUTE dbo.DatabaseIntegrityCheck
@Databases = 'USER_DATABASES',
@CheckCommands = 'CHECKDB';
```

✅ B) Medium DBs (100–500 GB)

Same pattern as small, just adjust schedules (off-peak window).

Daily fast check:

```
EXECUTE dbo.DatabaseIntegrityCheck
@Databases = 'USER_DATABASES',
@CheckCommands = 'CHECKDB',
@PhysicalOnly = 'Y';
```

Weekly full check:

```
EXECUTE dbo.DatabaseIntegrityCheck
@Databases = 'USER_DATABASES',
@CheckCommands = 'CHECKDB';
```

📌 *Schedule Full CHECKDB on Sundays / off-business hours.*

✅ C) Very Large DBs (500GB – 1TB+)

Real-world best practice: **Do NOT run FULL CHECKDB on production** unless you have a massive maintenance window.

🟢 On Production (DAILY):

```
EXECUTE dbo.DatabaseIntegrityCheck
@Databases = 'VeryLargeDB',
@CheckCommands = 'CHECKDB',
@PhysicalOnly = 'Y';
```

🔵 On DR / RESTORE Server (WEEKLY FULL):

1. Restore FULL backup to DR
2. Run:
   EXECUTE dbo.DatabaseIntegrityCheck
   @Databases = 'VeryLargeDB',
   @CheckCommands = 'CHECKDB';

✓ This avoids:

- 100GB+ snapshot tempdb pressure
- IO log thrashing
- CPU spikes
- Long blocking on prod

📌 3. When to use Ola Hallengren Parameters

Parameter	Use Case
@PhysicalOnly = 'Y'	Fast corruption detection, daily checks
@LogToTable = 'Y'	Saves output to CommandLog *
@Databases = 'SYSTEM_DATABASES,USER_DATABASES'	Full coverage

📌 Make sure Ola logging is ON **(Recommended)**
@LogToTable = 'Y';

📌 4. Scheduling Recommendation

Job	Frequency	Window
Daily CHECKDB (PHYSICAL_ONLY)	Mon–Sat	1–2 AM
Weekly FULL CHECKDB	Sunday	Long window
DR Weekly FULL CHECKDB for VLDB	Sunday	On restore server

📌 5. Advantages / Disadvantages (Reality)

Type	Pros	Cons
PHYSICAL_ONLY	Very fast, minimal IO	Misses logical corruption
FULL CHECKDB on prod	Best integrity validation	Heavy CPU/IO, tempdb pressure
Full CHECKDB on DR	Zero prod impact	Requires DR + restore automation

📌 6. Optional (Should I generate this for you?)

I can also provide:

✅ SQL Agent Job XML (importable)
✅ DR automation script → **restore latest backup + run CHECKDB + email results**
✅ Email notifications or Operator alerts (since you prefer **Option B earlier**)

DBCC CHECKDB with Ola Hallengren — Always On Availability Groups:

Below is a ready-to-run, production-grade package for running **FULL DBCC CHECKDB** for Very Large Databases (VLDBs) using **Ola Hallengren**, offloaded to a **readable secondary** in an **Always On Availability Group (AG)**, plus daily PHYSICAL_ONLY checks on primary. It includes safety checks, logging, operator notifications (SQL Agent operator), schedules, and tips for failover handling.

Executive summary (one-line)
Run daily PHYSICAL_ONLY on primary; run weekly **FULL** DBCC CHECKDB on a readable, synchronized AG secondary using Ola Hallengren; if no suitable secondary is available, alert and skip to avoid impacting production.

✅ **Pre-reqs & assumptions**

- Ola Hallengren Maintenance Solution is installed on the instance(s). (DatabaseIntegrityCheck procedure present in the dbo schema of the instance you run jobs on.)
- You have an SQL Agent operator configured (e.g., DBA_Operator) and Database Mail enabled.
- The AG secondary is configured as **Readable Secondary = Yes (Read-intent or All)**.
- You have permission to create SQL Agent jobs on the secondary.
- Logging will be written into Ola's command log tables (default dbo.CommandLog, dbo.CommandExecute) or your own DBA log (option shown).
- You accept that FULL CHECKDB on secondary still consumes CPU/I/O on that server — ensure sizing and windows are correct.

Design overview

1. **Primary Instance**
   - Daily job: PHYSICAL_ONLY via Ola for all user DBs (fast smoke test).
   - Alert if PHYSICAL_ONLY finds errors.
2. **AG Secondary (Readable)**
   - Weekly job (or configurable frequency): full DBCC CHECKDB using Ola on VLDB(s).
   - The job first verifies the local replica is **READABLE** and **SYNCHRONIZED** (or acceptable sync state you choose).
   - If not readable/synchronized, job logs/skips and notifies operator.
3. **Fallback / Alerting**
   - If no secondary is available for full CHECKDB, operator is notified so you can schedule a manual restore-based CHECKDB on a separate server.

Recommended schedule

- **Daily (Primary)**: PHYSICAL_ONLY — 02:00 AM local.
- **Weekly (Secondary)**: FULL CHECKDB — Sunday 03:00 AM local (stagger if multiple VLDBs).
- If multiple VLDBs: stagger across different nights to limit I/O on the secondary.

Scripts
Replace values like YourVGDB, YourAGName, operator name, server names, and schedules to match your environment. Run primary jobs on primary; full-check job on secondary.

1) Daily PHYSICAL_ONLY on Primary (Ola)
Create SQL Agent job on primary. This runs quickly and detects many physical issues.

```
-- Daily PHYSICAL_ONLY job (run on Primary)
EXECUTE dbo.DatabaseIntegrityCheck
  @Databases = 'USER_DATABASES',
  @CheckCommands = 'CHECKDB',
  @PhysicalOnly = 'Y',
  @LogToTable = 'Y',
  @Checksum = 'Y';  -- if you have page checksums enabled (recommended)
```

Notes

- Use Ola parameter @LogToTable = 'Y' so results appear in Ola's CommandLog.
- Schedule daily at 02:00 AM. Configure job notification to your operator on failure.

2) Weekly FULL CHECKDB on Readable AG Secondary (T-SQL job step)

Create this job on the **readable AG secondary** instance. It:

1. Validates the database is part of an AG and the local replica state is readable and synchronized.
2. If OK, calls Ola DatabaseIntegrityCheck for that specific DB (FULL).
3. Logs success or error to a DBA log table and notifies operator on failure or on skip.

Full job T-SQL step (single step content) — replace placeholders:

```
/* Job Step: Run FULL CHECKDB on readable AG secondary for one VLDB */
SET NOCOUNT ON;

DECLARE
   @DBName SYSNAME = N'YourVLDB',          -- <-- change DB name
   @AGName SYSNAME = N'YourAGName',        -- <-- change AG name (optional)
   @Operator SYSNAME = N'DBA_Operator',    -- <-- change operator
   @Result NVARCHAR(MAX) = N'',
   @ReplicaState INT,
   @Readable VARCHAR(20);

-- Function: check replica state & readability for this DB on local instance
SELECT TOP (1)
   @ReplicaState = ars.synchronization_state,        -- 1=NOT_SYNCHRONIZED, 2=SYNCHRONIZING, 3=SYNCHRONIZED (see dm view), etc.
   @Readable = rs.secondary_role_allow_connections_desc
FROM sys.availability_databases_cluster adc
JOIN sys.dm_hadr_database_replica_states ars
   ON adc.group_database_id = ars.group_database_id
JOIN sys.availability_replicas ar
   ON ars.replica_id = ar.replica_id
JOIN sys.availability_groups ag
   ON ar.group_id = ag.group_id
JOIN sys.availability_replicas_cluster rc
   ON rc.replica_server_name = CONVERT(sysname, SERVERPROPERTY('ServerName'))
JOIN sys.dm_hadr_availability_replica_states rs
   ON rs.replica_id = ar.replica_id
WHERE adc.database_name = @DBName
 AND ag.name = @AGName;

-- Validate: must be SYNCHRONIZED and secondary must allow read access
IF @ReplicaState IS NULL
BEGIN
   SET @Result = N'Error: database not part of AG or AG not found on this instance.';
   INSERT INTO DBA.dbo.DBA_CheckDB_Log(DatabaseName, CheckType, ResultText)
    VALUES (@DBName, N'AG_FULL_CHECKDB', @Result);
   -- notify operator
   EXEC msdb.dbo.sp_notify_operator @name = @Operator, @subject = 'CHECKDB SKIPPED - DB not part of AG', @message = @Result;
   THROW 51000, @Result, 1;
END

-- Check synchronization and readability
-- synchronization_state: 1=NOT SYNCHRONIZING, 2=SYNCHRONIZING, 3=SYNCHRONIZED
IF @ReplicaState <> 3
BEGIN
   SET @Result = N'AG replica is not synchronized (state=' + COALESCE(CONVERT(NVARCHAR(10), @ReplicaState), N'NULL') + '). Skipping
FULL CHECKDB.';
```

```
    INSERT INTO DBA.dbo.DBA_CheckDB_Log(DatabaseName, CheckType, ResultText)
     VALUES (@DBName, N'AG_FULL_CHECKDB', @Result);

    EXEC msdb.dbo.sp_notify_operator @name = @Operator, @subject = 'CHECKDB SKIPPED - Replica not synchronized', @message =
@Result;
    RETURN;  -- exit gracefully (job step success). Change to THROW if you want job to fail.
END

IF @Readable NOT IN ('READ_ONLY','ALL')
BEGIN
    SET @Result = N'AG replica is not configured as readable secondary (SecondaryRoleAllowConnections=' + ISNULL(@Readable,'NULL') + ').
Skipping FULL CHECKDB.';
    INSERT INTO DBA.dbo.DBA_CheckDB_Log(DatabaseName, CheckType, ResultText)
     VALUES (@DBName, N'AG_FULL_CHECKDB', @Result);

    EXEC msdb.dbo.sp_notify_operator @name = @Operator, @subject = 'CHECKDB SKIPPED - Replica not readable', @message = @Result;
    RETURN;
END

-- If we get here, safe to run FULL CHECKDB on this readable, synchronized secondary.
BEGIN TRY
    EXECUTE dbo.DatabaseIntegrityCheck
       @Databases = @DBName,
       @CheckCommands = 'CHECKDB',
       @PhysicalOnly = 'N',
       @LogToTable = 'Y',
       @OutputFileDirectory = N'\\some\share\CHECKDB_Results\',  -- optional: track output
       @OutputFileName = @DBName + '_CHECKDB_' + REPLACE(CONVERT(VARCHAR(19), GETDATE(), 120), ':', '-') + '.txt';
    SET @Result = N'OK - FULL CHECKDB completed';
    INSERT INTO DBA.dbo.DBA_CheckDB_Log(DatabaseName, CheckType, ResultText)
     VALUES (@DBName, N'AG_FULL_CHECKDB', @Result);
END TRY
BEGIN CATCH
    SET @Result = 'ERROR: ' + ERROR_MESSAGE();
    INSERT INTO DBA.dbo.DBA_CheckDB_Log(DatabaseName, CheckType, ResultText)
     VALUES (@DBName, N'AG_FULL_CHECKDB', @Result);
    -- Notify operator and fail the step so SQL Agent notifies automatically
    EXEC msdb.dbo.sp_notify_operator @name = @Operator, @subject = 'CHECKDB FAILED on AG Secondary', @message = @Result;
    THROW;
END CATCH;
```

Important notes for the step above

- sp_notify_operator is a convenience wrapper many shops add. If you don't have it, use EXEC msdb.dbo.sp_send_dbmail or configure job notification to the operator in the job properties.
- The script **checks** synchronization_state and secondary_role_allow_connections_desc. Adjust acceptance criteria if you allow SYNCHRONIZING or different read behavior.
- If you prefer the job to **fail** when not synchronized (so you get an immediate escalation), replace the RETURN with a THROW or RAISERROR to generate job failure and operator notification.

3) Multi-VLDB / Multi-Replica pattern (run multiple DBs on one secondary)

If you have many VLDBs, create one job per DB on different secondaries or stagger runs. Sample loop for several DBs:

```
DECLARE @DB SYSNAME;
DECLARE dbcur CURSOR LOCAL FAST_FORWARD FOR
    SELECT name FROM sys.databases
    WHERE name IN ('VLDB1','VLDB2','VLDB3');
```

```
OPEN dbcur;
FETCH NEXT FROM dbcur INTO @DB;
WHILE @@FETCH_STATUS = 0
BEGIN
   -- call DatabaseIntegrityCheck inside TRY/CATCH as above for each @DB
   FETCH NEXT FROM dbcur INTO @DB;
END
CLOSE dbcur;
DEALLOCATE dbcur;
```

Be careful: running multiple full CHECKDBs concurrently on the same secondary may saturate disk/CPU.

4) Failover & maintenance considerations

- **Planned failover**: if you plan to failover the AG secondary that runs CHECKDB, ensure you re-deploy or move the scheduled job to the new secondary (or use a central automation that can target the current readable secondary).
- **Unplanned failover**: if the secondary used for CHECKDB fails or becomes primary, the job should be disabled on that server and enabled on the new readable secondary. Consider using a central orchestrator (PowerShell/Runbook) that detects the current readable secondary and runs the job there.
- **Automation recommendation**: Use PowerShell (or an automation runbook) that:
    1. Queries AG metadata (sys.dm_hadr_availability_replica_states, sys.dm_hadr_database_replica_states) to find the preferred readable secondary.
    2. Uses Invoke-Sqlcmd to remotely trigger the Ola DatabaseIntegrityCheck or enable a job on that server. This avoids editing jobs after each failover.

5) Logging & alerts (operator integration)

- Use Ola's @LogToTable = 'Y' to record details in dbo.CommandLog and dbo.CommandExecute. Monitor those tables for failures.
- Configure SQL Agent job notifications: Job Properties → Notifications → When the job fails → Notify Operator DBA_Operator.
- Optionally, add an SQL Agent Alert for error numbers 824, 823, 825, 898, and 601 which often indicate I/O/consistency problems:

```
EXEC msdb.dbo.sp_add_alert @name=N'SQL_Error_824', @message_id=824, @severity=0, @enabled=1;
EXEC msdb.dbo.sp_add_notification @alert_name=N'SQL_Error_824', @operator_name=N'DBA_Operator', @notification_method = 1;
```

6) Practical tuning & best practices (AG + Ola)

- **Readable Secondary must be healthy** — ensure backups, integrity, and monitoring are applied to the secondary as well.
- **Tempdb & IO** — even on secondary, full CHECKDB reads everything and can stress storage. Ensure secondary has headroom.
- **Stagger VLDB jobs** — run only one VLDB full-check at a time on a given server.
- **Page checksums enabled** (PAGE_VERIFY = CHECKSUM) — this helps detect corruption earlier.
- **Have tested backups** — even if CHECKDB runs clean, you must have tested restores. CHECKDB does not replace backup strategy.
- **Use Ola logging & history** — monitor command duration and trends to detect growth in run times (indicates increasing DB churn/size).
- **Snapshot creation on secondary** — when running full CHECKDB on secondary, it still creates a snapshot if run against the database copy; the secondary will create a sparse snapshot — ensure disk space on the secondary for snapshots.

7) Example: PowerShell Orchestrator (simple pattern)

A short PowerShell sketch to find a readable secondary and call a remote job (conceptual — customize for your environment):

```
# PSEUDO-CODE (not production ready)
$agName = "YourAGName"
$targetDb = "YourVLDB"
```

```
# query primary for replica info (use Invoke-Sqlcmd or SMO)

$query = @"
SELECT replica_server_name
FROM sys.availability_replicas ar
JOIN sys.availability_groups ag ON ar.group_id = ag.group_id
JOIN sys.dm_hadr_availability_replica_states rs ON rs.replica_id = ar.replica_id
WHERE ag.name = '$agName' AND rs.secondary_role_allow_connections_desc IN ('READ_ONLY','ALL')
"@

$replicas = Invoke-Sqlcmd -ServerInstance "PrimaryHost" -Query $query
# choose one replica from $replicas (prefer SYNCHRONIZED)
# then remotely run: Start-Job or invoke-sqlcmd to kick off the job on replica
Invoke-Command -ComputerName $replicas.replica_server_name -ScriptBlock {
  Import-Module SqlServer
  Invoke-Sqlcmd -ServerInstance "." -Database "msdb" -Query "EXEC msdb.dbo.sp_start_job N'AG_FULL_CHECKDB_YourVLDB'"
}
```

Use this approach to centralize scheduling and to avoid changing jobs after failover.

8) Post-run actions & remediation steps on detection of corruption

1. If Ola / CHECKDB finds corruption on the secondary, capture full output (ALL_ERRORMSGS) and escalate to DBA/storage team.
2. Validate backups (test restore the latest good backup to an isolated server and run CHECKDB there).
3. If backup not available, consult recovery options — REPAIR_ALLOW_DATA_LOSS only as last resort and after full backups are taken.
4. Investigate root cause: SAN, HBA/driver, storage firmware, networked storage issues, OS-level errors in event logs.

9) Quick checklist to deploy

1. Ensure readable, synchronized AG secondary exists and has sufficient disk/IO.
2. Install Ola Hallengren on the secondary if not present (the procedure executes locally).
3. Create DBA logging table (DBA.dbo.DBA_CheckDB_Log) or rely on Ola CommandLog.
4. Create SQL Agent job on primary for daily PHYSICAL_ONLY (Ola call).
5. Create SQL Agent job on secondary for weekly FULL CHECKDB (use script above). Set job notifications to DBA_Operator.
6. Test the job manually during a maintenance window; verify logs and mail notifications.
7. Implement runbook describing steps to follow when notification arrives (who to call, how to capture output, how to escalate).

10) Example Ola call (single command) for full check of one DB

If you prefer the single direct call (what the job executes):

```
EXECUTE dbo.DatabaseIntegrityCheck
   @Databases = 'YourVLDB',
   @CheckCommands = 'CHECKDB',
   @PhysicalOnly = 'N',
   @LogToTable = 'Y',
   @OutputFileDirectory = N'\\backup-share\CHECKDB_Outputs\YourVLDB\';
```

When running DBCC CHECKDB from a Secondary replica, there are **two valid safety models:**

✅ **Model 1 — STRICT SAFETY (Most Common & Recommended)**

Job runs only when:

Check	Required
SQL Server is **Secondary Replica**	✅
Replica role is **Readable Secondary**	✅
Replica sync state is **SYNCHRONIZED**	✅
Database is **NOT SUSPENDED**	✅

Behavior:

If any condition fails → job **skips automatically** and logs: *"CHECKDB skipped — replica not healthy."*

➡️ *Goal: Zero risk, never runs in questionable state.*

✅ **Model 2 — FLEX SAFETY**

Job runs if:

Check	Required
SQL Server is **Secondary Replica**	✅
Database is **READABLE**	✅

It **does NOT require SYNCHRONIZED state**.

Behavior:

Will still run if AG briefly falls behind.

➡️ *Goal: CHECKDB always runs weekly — even if sync is behind a little.*

📌 **The sql file will contain:**

Component	Location
Job 1: DBCC CHECKDB PHYSICAL_ONLY (Daily)	**Primary** replica
Job 2: DBCC CHECKDB FULL (Weekly)	**Secondary** replica
Even if AG role flips	Jobs auto-detect and auto-skip
Includes @LogToTable = 'Y'	using Ola Hallengren
Fully idempotent	Safe to re-run
Includes TRY/CATCH + RAISERROR + PRINT output	For operators

🛠 **SQL Agent Job Safety Logic inside the script**

Example of what will be inside the job step (simplified):

```
IF (SELECT primary_replica FROM sys.dm_hadr_availability_group_states
   WHERE group_id = (SELECT group_id FROM sys.databases WHERE name = 'MyDB'))
   = @@SERVERNAME
BEGIN
   PRINT 'I am PRIMARY, skipping FULL CHECKDB';
   RETURN;
END
```

AG DB List preference

Option	Meaning
ALL	Run CHECKDB for all AG databases
LIST	I will specify a list like 'DB1,DB2,DB3'

SafetyModel: 1 or 2
Databases: ALL or LIST (if LIST, provide DB names)

SafetyModel: 1
Databases: ALL

✅ SafetyModel

Option	Description
1 (Strict Safety – Recommended)	Run FULL CHECKDB on secondary **only if:** Secondary role + Readable + Synchronized + Not Suspended
2 (Flexible Safety)	Run FULL CHECKDB on secondary **as long as it is Secondary & Readable**, even if not fully synchronized

✅ Choose Databases

Option	Description
ALL	Run CHECKDB on **all AG databases**
LIST	You will provide a list, e.g. 'DB1,DB2,DB3'

Below is a single, **ready-to-run .sql file** that:

- Creates a logging DB/table (DBA.dbo.DBA_CheckDB_Log) if not present
- Creates a SQL Agent **Operator** (DBA_Operator) (edit email to your real address)
- Creates **Job A**: Daily PHYSICAL_ONLY (Ola Hallengren) — meant for Primary (but safe to run anywhere)
- Creates **Job B**: Weekly **FULL CHECKDB** (Ola Hallengren) on a **readable AG secondary** for the DB list you requested (DB1, DB2) using **SafetyModel 2** logic (runs if local replica is SECONDARY and is readable)
- Jobs are **idempotent** (safe to re-run the script).
- Jobs notify the Operator on failure (SQL Agent job-level notification).

Before you run

1. Ensure **Ola Hallengren** (DatabaseIntegrityCheck) is installed on the instances where you will run these jobs.
2. Edit the placeholders near the top of the script:
   - @OperatorEmail — change to your team email (important to receive notifications)
   - @DailyStartTime / @WeeklyStartTime — adjust start times if needed
   - @DBList — currently 'DB1','DB2' (already set per your request)
   - If you use a different operator name, change @OperatorName.
3. Run the file on the **Primary** instance to create the Primary daily job, and run the same file (or copy) on the **intended AG Secondary** instance to create the Secondary weekly job. The Secondary job has safety checks and will skip if the local replica is not SECONDARY+readable.

```
/* ------------------------------------------------------------------------
  READY-TO-RUN: DBCC CHECKDB (Ola Hallengren) - Primary Daily PHYSICAL_ONLY
              + AG Secondary Weekly FULL CHECKDB (SafetyModel 2)
  Notes: Update @OperatorEmail and times below before execution.
  Run this script on the Primary to create the Daily job.
  Run this same script on the intended AG Secondary to create the Weekly job.
  ------------------------------------------------------------------------ */
SET NOCOUNT ON;
GO
/* ---------------------------
  CONFIGURATION - EDIT THESE
  --------------------------- */
DECLARE @OperatorName SYSNAME = N'DBA_Operator';
DECLARE @OperatorEmail NVARCHAR(256) = N'dba-team@yourcompany.com'; -- << CHANGE THIS
DECLARE @DailyStartTime INT = 20000;   -- 02:00:00 (HHMMSS as integer) - Daily Physical_only
DECLARE @WeeklyStartTime INT = 30000;  -- 03:00:00 (HHMMSS) - Weekly FULL on secondary
DECLARE @DBList NVARCHAR(MAX) = N'''DB1'',''DB2''';  -- DB list for FULL CHECK on secondary - change as required
/* ---------------------------
  END CONFIG
  --------------------------- */

-- 1) Create DBA database and logging table if not exists
IF DB_ID('DBA') IS NULL
BEGIN
   PRINT 'Creating DBA database...';
   CREATE DATABASE DBA;
   -- Optionally set file locations - left default for simplicity
END
GO
```

```
USE DBA;
GO

IF OBJECT_ID('dbo.DBA_CheckDB_Log','U') IS NULL
BEGIN
   PRINT 'Creating DBA.dbo.DBA_CheckDB_Log table...';
   CREATE TABLE dbo.DBA_CheckDB_Log (
      LogID INT IDENTITY PRIMARY KEY,
      DatabaseName SYSNAME NOT NULL,
      CheckType VARCHAR(30) NOT NULL,
      RunDate DATETIME2 NOT NULL DEFAULT SYSUTCDATETIME(),
      ResultText NVARCHAR(MAX) NULL
   );
END
GO

-- 2) Create Operator in msdb if not exists (so SQL Agent notifications work)
USE msdb;
GO
IF NOT EXISTS (SELECT 1 FROM msdb.dbo.sysoperators WHERE name = @OperatorName)
BEGIN
   PRINT 'Creating SQL Agent Operator: ' + @OperatorName;
   EXEC msdb.dbo.sp_add_operator
      @name = @OperatorName,
      @enabled = 1,
      @email_address = @OperatorEmail;
END
ELSE
   PRINT 'Operator exists: ' + @OperatorName;
GO
/* ---------------------------
  JOB A: Daily PHYSICAL_ONLY (All user DBs)
  - Uses Ola Hallengren DatabaseIntegrityCheck with @PhysicalOnly='Y'
  - Schedule: Daily at @DailyStartTime
  --------------------------- */
DECLARE @JobNameA SYSNAME = N'DBCC_CHECKDB_Daily_PHYSICAL_ONLY_Ola';
DECLARE @Owner SYSNAME = SUSER_SNAME();

-- Delete job if exists (idempotent)
IF EXISTS (SELECT 1 FROM msdb.dbo.sysjobs WHERE name = @JobNameA)
BEGIN
   PRINT 'Cleaning up existing job: ' + @JobNameA;
   EXEC msdb.dbo.sp_delete_job @job_name = @JobNameA;
END
-- Create job
EXEC msdb.dbo.sp_add_job
   @job_name = @JobNameA,
   @owner_login_name = @Owner,
   @enabled = 1,
   @description = N'Daily DBCC CHECKDB (PHYSICAL_ONLY) for all user databases using Ola Hallengren',
   @notify_level_email = 2,  -- notify on failure
   @notify_email_operator_name = @OperatorName;
GO
```

```
-- Add job step: call Ola DatabaseIntegrityCheck (physical only)
EXEC msdb.dbo.sp_add_jobstep
   @job_name = @JobNameA,
   @step_name = N'Run PHYSICAL_ONLY via Ola (USER_DATABASES)',
   @subsystem = N'TSQL',
   @command = N'
-- Daily PHYSICAL_ONLY check (Ola Hallengren)
SET NOCOUNT ON;
BEGIN TRY
   EXECUTE dbo.DatabaseIntegrityCheck
      @Databases = ''USER_DATABASES'',
      @CheckCommands = ''CHECKDB'',
      @PhysicalOnly = ''Y'',
      @LogToTable = ''Y'';
   INSERT INTO DBA.dbo.DBA_CheckDB_Log(DatabaseName, CheckType, ResultText)
   VALUES (''ALL_USER_DBS'', ''PHYSICAL_ONLY'', ''OK'');
END TRY
BEGIN CATCH
   INSERT INTO DBA.dbo.DBA_CheckDB_Log(DatabaseName, CheckType, ResultText)
   VALUES (''ALL_USER_DBS'', ''PHYSICAL_ONLY'', ERROR_MESSAGE());
   THROW;
END CATCH;
',
   @on_success_action = 1,  -- Quit with success
   @on_fail_action = 2;     -- Quit with failure
GO
-- Add schedule: daily at @DailyStartTime
EXEC msdb.dbo.sp_add_schedule
   @schedule_name = N'Sched_Daily_PHYSICAL_ONLY_Ola',
   @freq_type = 4,          -- daily
   @freq_interval = 1,
   @active_start_time = @DailyStartTime;
EXEC msdb.dbo.sp_attach_schedule
   @job_name = @JobNameA,
   @schedule_name = N'Sched_Daily_PHYSICAL_ONLY_Ola';
EXEC msdb.dbo.sp_add_jobserver
   @job_name = @JobNameA,
   @server_name = @@SERVERNAME;
GO
PRINT 'Job created: ' + @JobNameA + ' with daily schedule.';
GO

/* ---------------------------
  JOB B: Weekly FULL CHECKDB on Readable AG Secondary (SafetyModel 2)
  - Runs full CHECKDB (Ola) for DBs in @DBList when local replica is:
      role_desc = ''SECONDARY'' AND replica allows read (READ_ONLY or ALL)
  - Does NOT require SYNCHRONIZED (SafetyModel 2)
  - Schedule: Weekly at @WeeklyStartTime
  --------------------------- */
DECLARE @JobNameB SYSNAME = N'AG_Secondary_Weekly_FULL_CHECKDB_Ola';
-- Remove existing job if present to support idempotence
IF EXISTS (SELECT 1 FROM msdb.dbo.sysjobs WHERE name = @JobNameB)
```

```
BEGIN
   PRINT 'Cleaning up existing job: ' + @JobNameB;
   EXEC msdb.dbo.sp_delete_job @job_name = @JobNameB;
END
-- Create the job (it is safe to create on Primary too; job contains local checks)
EXEC msdb.dbo.sp_add_job
   @job_name = @JobNameB,
   @owner_login_name = @Owner,
   @enabled = 1,
   @description = N'Weekly FULL DBCC CHECKDB (Ola) for AG DB list on readable secondary (SafetyModel 2)',
   @notify_level_email = 2,
   @notify_email_operator_name = @OperatorName;
GO
/* Job step: For each DB in the provided list, check local replica role and readability.
   If role = SECONDARY and replica allows read, run full CHECKDB via Ola for that DB.
   Otherwise, log a SKIPPED message. */

DECLARE @StepCommand NVARCHAR(MAX) = N'
SET NOCOUNT ON;

DECLARE @DB SYSNAME;
DECLARE @db_list TABLE (name SYSNAME);
INSERT INTO @db_list(name) VALUES (:dbvals); -- placeholder: replaced when creating step

DECLARE cur CURSOR LOCAL FAST_FORWARD FOR SELECT name FROM @db_list;
OPEN cur;
FETCH NEXT FROM cur INTO @DB;
WHILE @@FETCH_STATUS = 0
BEGIN
   DECLARE @role_desc SYSNAME = NULL;
   DECLARE @allow_connections_desc NVARCHAR(60) = NULL;
   DECLARE @res NVARCHAR(MAX) = N'''';

   -- Check whether this database is part of an AG and whether local replica is SECONDARY
   SELECT TOP(1)
      @role_desc = drs.role_desc
   FROM sys.dm_hadr_database_replica_states drs
   JOIN sys.availability_databases_cluster adc ON drs.group_database_id = adc.group_database_id
   WHERE adc.database_name = @DB AND drs.is_local = 1;

   -- Check whether local replica configuration allows read connections (READ_ONLY or ALL)
   SELECT TOP(1)
      @allow_connections_desc = ars.secondary_role_allow_connections_desc
   FROM sys.dm_hadr_availability_replica_states ars
   JOIN sys.availability_replicas ar ON ars.replica_id = ar.replica_id
   JOIN sys.availability_groups ag ON ar.group_id = ag.group_id
   JOIN sys.availability_databases_cluster adc ON ag.group_id = adc.group_id
   WHERE adc.database_name = @DB AND ars.is_local = 1;

   IF @role_desc IS NULL
   BEGIN
      SET @res = N''SKIPPED - DB not part of AG on this instance or AG metadata not present.'';
   INSERT INTO DBA.dbo.DBA_CheckDB_Log(DatabaseName, CheckType, ResultText) VALUES (@DB, N''AG_FULL_CHECKDB'',
@res);
```

```
 -- go next DB
      FETCH NEXT FROM cur INTO @DB;
      CONTINUE;
   END

   IF UPPER(ISNULL(@role_desc,'')) <> ''SECONDARY''
   BEGIN
      SET @res = N''SKIPPED - Local replica is not SECONDARY (role=''+ISNULL(@role_desc,''NULL'')+'')'';
      INSERT INTO DBA.dbo.DBA_CheckDB_Log(DatabaseName, CheckType, ResultText) VALUES (@DB, N''AG_FULL_CHECKDB'',
@res);
      FETCH NEXT FROM cur INTO @DB;
      CONTINUE;
   END

   IF UPPER(ISNULL(@allow_connections_desc,'')) NOT IN (''READ_ONLY'',''ALL'')
   BEGIN
      SET @res = N''SKIPPED - Local replica does not allow read connections
(secondary_role_allow_connections_desc=''+ISNULL(@allow_connections_desc,''NULL'')+'')'';
      INSERT INTO DBA.dbo.DBA_CheckDB_Log(DatabaseName, CheckType, ResultText) VALUES (@DB, N''AG_FULL_CHECKDB'',
@res);
      FETCH NEXT FROM cur INTO @DB;
      CONTINUE;
   END

   -- At this point: SafetyModel 2 conditions satisfied: local = SECONDARY AND readable -> run FULL CHECKDB
   BEGIN TRY
      EXECUTE dbo.DatabaseIntegrityCheck
         @Databases = @DB,
         @CheckCommands = ''CHECKDB'',
         @PhysicalOnly = ''N'',
         @LogToTable = ''Y'';
      INSERT INTO DBA.dbo.DBA_CheckDB_Log(DatabaseName, CheckType, ResultText) VALUES (@DB, N''AG_FULL_CHECKDB'',
N''OK'');
   END TRY
   BEGIN CATCH
      INSERT INTO DBA.dbo.DBA_CheckDB_Log(DatabaseName, CheckType, ResultText) VALUES (@DB, N''AG_FULL_CHECKDB'',
ERROR_MESSAGE());
      -- Rethrow to cause job step failure so SQL Agent notifies operator (optional)
      THROW;
   END CATCH;

   FETCH NEXT FROM cur INTO @DB;
END
CLOSE cur;
DEALLOCATE cur;
';

-- Replace placeholder :dbvals with actual list of DBs (escaped single quotes)
-- Build the actual command by replacing the placeholder
DECLARE @dbvals NVARCHAR(MAX) = @DBList; -- e.g. '''DB1'',''DB2'''
-- Build final step command replacing token
SET @StepCommand = REPLACE(@StepCommand, ':dbvals', @dbvals);
```

```
-- Add job step
EXEC msdb.dbo.sp_add_jobstep
   @job_name = @JobNameB,
   @step_name = N'AG Secondary FULL CHECKDB for configured DB list',
   @subsystem = N''TSQL'',
   @command = @StepCommand,
   @on_success_action = 1,  -- Quit with success
   @on_fail_action = 2;     -- Quit with failure
GO

-- Add weekly schedule (Sunday by default)
EXEC msdb.dbo.sp_add_schedule
   @schedule_name = N'Sched_Weekly_AG_Secondary_FULL_CHECK_Ola',
   @freq_type = 8,           -- weekly
   @freq_interval = 1,
   @active_start_time = @WeeklyStartTime;
EXEC msdb.dbo.sp_attach_schedule
   @job_name = @JobNameB,
   @schedule_name = N'Sched_Weekly_AG_Secondary_FULL_CHECK_Ola';

-- Attach job to server
EXEC msdb.dbo.sp_add_jobserver
   @job_name = @JobNameB,
   @server_name = @@SERVERNAME;
GO

PRINT 'Job created: ' + @JobNameB + ' with weekly schedule.';

-- Final message
PRINT 'Script completed. Review placeholder values at top (OperatorEmail, times, DBList).';
GO
```

Script to automatically detect whether it's running on Primary vs Secondary and create only the appropriate job:

Below is a **single ready-to-run .sql file** that:

- Automatically **detects** whether the instance is a Primary and/or a Secondary for the specified AG databases and **only creates the relevant jobs** for that role.
  - If the instance is Primary for any of the target DBs → it creates the **Daily PHYSICAL_ONLY** job (one job for all user DBs).
  - If the instance is a Secondary for any of the target DBs → it creates **one Weekly FULL CHECKDB job per DB** (safe for SafetyModel 2: runs when local replica = SECONDARY AND readable).
  - If the instance is both Primary for some DBs and Secondary for others (multi-replica host), it will create both sets of jobs as applicable.
- Implements **staggering** of FULL CHECKDB jobs across a configurable number of nights (default 3 nights). Each DB on the secondary will have its own SQL Agent job scheduled on a different day of the week in a round-robin fashion across the stagger window.
- Uses **Ola Hallengren** DatabaseIntegrityCheck calls (@LogToTable = 'Y'). You must have Ola installed on the instance(s) where jobs run.
- Is **idempotent** — safe to run multiple times; existing jobs are cleaned up and recreated.
- Uses **SafetyModel 2**: run FULL only if local replica is SECONDARY and replica allows read (READ_ONLY or ALL). It does **not** require SYNCHRONIZED.

How to use

1. Edit the configuration section near the top:
   - @OperatorEmail — set to your email.
   - @DBListCSV — comma-separated DB names (e.g., DB1,DB2,DB3). Script will trim spaces.
   - @DailyStartTime / @WeeklyStartTime — times in HHMMSS integer (e.g., 020000 for 02:00:00).
   - @StaggerDays — number of distinct nights to stagger across (default 3). Must be 1..7.
2. Run this file on the instance(s). You can run the same file on both primary and secondary; the script auto-detects role(s) and creates the appropriate jobs only.
3. Verify jobs in SQL Agent → Jobs, and test by running them manually once.

The script

```
/* ********************************************************************
  AUTO-ROLE DBCC CHECKDB (.sql)
  - Creates:
    * DBA.dbo.DBA_CheckDB_Log (if not exists)
    * SQL Agent Operator (if not exists)
    * Daily PHYSICAL_ONLY job (Primary instances) - single job for all user DBs
    * Weekly FULL CHECKDB jobs (Secondary instances) - one job per DB, staggered across nights
  - Uses Ola Hallengren DatabaseIntegrityCheck (must be installed)
  - SafetyModel 2: FULL runs if local replica ROLE = SECONDARY AND replica allows read (READ_ONLY or ALL)
  - Idempotent: deletes existing jobs of same names before creating them
  - Edit configuration section below before running
  ******************************************************************** */
SET NOCOUNT ON;
GO
/* ----------------------------
  CONFIG - EDIT THESE VALUES
  ---------------------------- */
DECLARE @OperatorName SYSNAME = N'DBA_Operator';
DECLARE @OperatorEmail NVARCHAR(256) = N'dba-team@yourcompany.com'; -- << CHANGE THIS
DECLARE @DBListCSV NVARCHAR(MAX) = N'DB1,DB2';   -- comma-separated DB names (no quotes)
```

```
DECLARE @DailyStartTime INT = 20000;   -- Daily physical_only run time (HHMMSS): 02:00:00 = 20000
DECLARE @WeeklyStartTime INT = 30000;  -- Weekly full CHECKDB start time (HHMMSS)
DECLARE @StaggerDays INT = 3;          -- number of nights to stagger full checks across (1..7). Default 3
/* ----------------------------
  END CONFIG
  ---------------------------- */

-- Validate @StaggerDays bounds
IF @StaggerDays < 1 OR @StaggerDays > 7
BEGIN
   RAISERROR('Invalid @StaggerDays value. Must be between 1 and 7.', 16, 1);
   RETURN;
END
/* ----------------------------
  Helper: create DBA database/logging table if not exists
  ---------------------------- */
IF DB_ID('DBA') IS NULL
BEGIN
   PRINT 'Creating DBA database...';
   CREATE DATABASE DBA;
END
GO

USE DBA;
GO

IF OBJECT_ID('dbo.DBA_CheckDB_Log','U') IS NULL
BEGIN
   PRINT 'Creating DBA.dbo.DBA_CheckDB_Log table...';
   CREATE TABLE dbo.DBA_CheckDB_Log (
      LogID INT IDENTITY PRIMARY KEY,
      DatabaseName SYSNAME NOT NULL,
      CheckType VARCHAR(30) NOT NULL,
      RunDate DATETIME2 NOT NULL DEFAULT SYSUTCDATETIME(),
      ResultText NVARCHAR(MAX) NULL
   );
END
GO

/* ----------------------------
  Create SQL Agent Operator if not exists
  ---------------------------- */
USE msdb;
GO

IF NOT EXISTS (SELECT 1 FROM msdb.dbo.sysoperators WHERE name = @OperatorName)
BEGIN
   PRINT 'Creating SQL Agent Operator: ' + @OperatorName;
   EXEC msdb.dbo.sp_add_operator
      @name = @OperatorName,
      @enabled = 1,
      @email_address = @OperatorEmail;
END
```

```
ELSE
   PRINT 'Operator exists: ' + @OperatorName;
GO

/* ----------------------------
  Parse @DBListCSV into table variable
  ---------------------------- */
DECLARE @DBTable TABLE (DBName SYSNAME NOT NULL, RowNum INT IDENTITY(1,1));
;WITH cte AS (
   SELECT LTRIM(RTRIM(value)) AS val
   FROM STRING_SPLIT(@DBListCSV, ',')
)
INSERT INTO @DBTable (DBName)
SELECT val FROM cte WHERE val <> '';
-- If no DBs provided -> stop
IF NOT EXISTS (SELECT 1 FROM @DBTable)
BEGIN
   RAISERROR('No databases specified in @DBListCSV. Edit the script and provide at least one DB name.', 16, 1);
   RETURN;
END

/* ----------------------------
  Determine roles for this instance
  - For each DB in list, check sys.dm_hadr_database_replica_states (role_desc & is_local)
  - For readability, we will create:
     - Primary job if the instance is PRIMARY for any of these DBs
     - Secondary jobs for DBs where instance is SECONDARY and allows read
  ---------------------------- */

-- Temp table to hold per-DB role state
DECLARE @RoleInfo TABLE (
   DBName SYSNAME PRIMARY KEY,
   IsLocalPrimary BIT DEFAULT 0,
   IsLocalSecondary BIT DEFAULT 0,
   SecondaryAllowConnectionsDesc NVARCHAR(60) NULL
);

DECLARE @d SYSNAME;
DECLARE curdb CURSOR LOCAL FAST_FORWARD FOR SELECT DBName FROM @DBTable;
OPEN curdb;
FETCH NEXT FROM curdb INTO @d;
WHILE @@FETCH_STATUS = 0
BEGIN
   -- Initialize row
   INSERT INTO @RoleInfo(DBName) VALUES (@d);

   -- Fill role info from dm_hadr view (if AG metadata present on this instance)
   UPDATE ri
   SET IsLocalPrimary = CASE WHEN drs.role_desc = 'PRIMARY' AND drs.is_local = 1 THEN 1 ELSE IsLocalPrimary END,
      IsLocalSecondary = CASE WHEN drs.role_desc = 'SECONDARY' AND drs.is_local = 1 THEN 1 ELSE IsLocalSecondary END
   FROM @RoleInfo ri
   LEFT JOIN sys.dm_hadr_database_replica_states drs
```

```
ON ri.DBName = (
            SELECT adc.database_name FROM sys.availability_databases_cluster adc
            WHERE adc.group_database_id = drs.group_database_id
          )
   WHERE ri.DBName = @d;

   -- Fill secondary allow connections description (if available)
   UPDATE ri
   SET SecondaryAllowConnectionsDesc = ars.secondary_role_allow_connections_desc
   FROM @RoleInfo ri
   LEFT JOIN sys.dm_hadr_availability_replica_states ars
      ON ars.is_local = 1
   LEFT JOIN sys.availability_replicas ar
      ON ars.replica_id = ar.replica_id
   LEFT JOIN sys.availability_groups ag
      ON ar.group_id = ag.group_id
   LEFT JOIN sys.availability_databases_cluster adc
      ON adc.group_id = ag.group_id
   WHERE ri.DBName = @d;
   -- Note: the above join is best-effort; if AG metadata not present, values remain null/0

   FETCH NEXT FROM curdb INTO @d;
END
CLOSE curdb;
DEALLOCATE curdb;

-- Check flags: isPrimaryInstance = exists any IsLocalPrimary = 1
DECLARE @isPrimaryInstance BIT = (CASE WHEN EXISTS (SELECT 1 FROM @RoleInfo WHERE IsLocalPrimary = 1) THEN 1 ELSE 0
END);
-- Secondary DBs list: those rows where IsLocalSecondary=1 and allow connections in (READ_ONLY, ALL)
DECLARE @SecondaryCount INT = (SELECT COUNT(1) FROM @RoleInfo WHERE IsLocalSecondary = 1 AND
UPPER(ISNULL(SecondaryAllowConnectionsDesc,'')) IN ('READ_ONLY','ALL'));

PRINT 'Instance role evaluation complete.';
PRINT 'IsPrimaryInstance=' + CONVERT(VARCHAR(1), @isPrimaryInstance) + ', SecondaryDBsCount=' +
CONVERT(VARCHAR(10), @SecondaryCount);

/* ----------------------------
  Create Primary job: Daily PHYSICAL_ONLY (if this instance is PRIMARY for any listed DB)
  - Name: DBCC_CHECKDB_Daily_PHYSICAL_ONLY_Ola
  - Uses Ola DatabaseIntegrityCheck @Databases='USER_DATABASES', @PhysicalOnly='Y'
  ---------------------------- */
IF @isPrimaryInstance = 1
BEGIN
   PRINT 'Creating Primary Daily PHYSICAL_ONLY job...';

   DECLARE @JobNameA SYSNAME = N'DBCC_CHECKDB_Daily_PHYSICAL_ONLY_Ola';
   DECLARE @Owner SYSNAME = SUSER_SNAME();

   IF EXISTS (SELECT 1 FROM msdb.dbo.sysjobs WHERE name = @JobNameA)
```

```
  BEGIN
     PRINT 'Deleting existing job: ' + @JobNameA;
     EXEC msdb.dbo.sp_delete_job @job_name = @JobNameA;
   END

   EXEC msdb.dbo.sp_add_job
     @job_name = @JobNameA,
     @owner_login_name = @Owner,
     @enabled = 1,
     @description = N'Daily DBCC CHECKDB (PHYSICAL_ONLY) for all user databases using Ola Hallengren',
     @notify_level_email = 2,
     @notify_email_operator_name = @OperatorName;

   EXEC msdb.dbo.sp_add_jobstep
     @job_name = @JobNameA,
     @step_name = N'Run PHYSICAL_ONLY via Ola (USER_DATABASES)',
     @subsystem = N'TSQL',
     @command = N'
SET NOCOUNT ON;
BEGIN TRY
   EXECUTE dbo.DatabaseIntegrityCheck
     @Databases = ''USER_DATABASES'',
     @CheckCommands = ''CHECKDB'',
     @PhysicalOnly = ''Y'',
     @LogToTable = ''Y'';
   INSERT INTO DBA.dbo.DBA_CheckDB_Log(DatabaseName, CheckType, ResultText)
     VALUES (''ALL_USER_DBS'', ''PHYSICAL_ONLY'', ''OK'');
END TRY
BEGIN CATCH
   INSERT INTO DBA.dbo.DBA_CheckDB_Log(DatabaseName, CheckType, ResultText)
     VALUES (''ALL_USER_DBS'', ''PHYSICAL_ONLY'', ERROR_MESSAGE());
   THROW;
END CATCH;
',
     @on_success_action = 1,
     @on_fail_action = 2;

   -- schedule: daily
   IF EXISTS (SELECT 1 FROM msdb.dbo.sysschedules WHERE name = N'Sched_Daily_PHYSICAL_ONLY_Ola')
   BEGIN
     EXEC msdb.dbo.sp_delete_schedule @schedule_name = N'Sched_Daily_PHYSICAL_ONLY_Ola';
   END

   EXEC msdb.dbo.sp_add_schedule
     @schedule_name = N'Sched_Daily_PHYSICAL_ONLY_Ola',
     @freq_type = 4,          -- daily
     @freq_interval = 1,
     @active_start_time = @DailyStartTime;

   EXEC msdb.dbo.sp_attach_schedule @job_name = @JobNameA, @schedule_name = N'Sched_Daily_PHYSICAL_ONLY_Ola';
   EXEC msdb.dbo.sp_add_jobserver @job_name = @JobNameA, @server_name = @@SERVERNAME;
```

```
 PRINT 'Primary job created: ' + @JobNameA;
END
ELSE
   PRINT 'Instance not primary for listed DBs: skipping creation of Primary daily job.';

/* ----------------------------
  Create Secondary jobs: Weekly FULL per DB, staggered across @StaggerDays
  - For each DB in @DBTable where this instance is local SECONDARY and readable (SafetyModel 2),
    create job named AG_SECONDARY_FULL_CHECKDB_<DB>
  - Schedule weekly on a weekday determined by round-robin across @StaggerDays starting Sunday
    (Sunday->Monday->... mapping via day bitmask)
  ---------------------------- */
IF @SecondaryCount > 0
BEGIN
   PRINT 'Creating Weekly FULL CHECKDB jobs on this Secondary (SafetyModel 2)...';

   -- Day bitmask mapping: Sunday=1, Mon=2, Tue=4, Wed=8, Thu=16, Fri=32, Sat=64
   DECLARE @DayBits TABLE (DayIndex INT, DayName NVARCHAR(10), DayBit INT);
   INSERT INTO @DayBits (DayIndex, DayName, DayBit)
   VALUES (1,'Sunday',1),(2,'Monday',2),(3,'Tuesday',4),(4,'Wednesday',8),(5,'Thursday',16),(6,'Friday',32),(7,'Saturday',64);

   -- Build list of DBs that are local secondary + readable
   DECLARE @EligibleDBs TABLE (DBName SYSNAME, RowNum INT IDENTITY(1,1));
   INSERT INTO @EligibleDBs (DBName)
   SELECT ri.DBName
   FROM @RoleInfo ri
   WHERE ri.IsLocalSecondary = 1 AND UPPER(ISNULL(ri.SecondaryAllowConnectionsDesc,'')) IN ('READ_ONLY','ALL');

   DECLARE @TotalEligible INT = (SELECT COUNT(1) FROM @EligibleDBs);
   PRINT 'Number of eligible DBs on this instance for FULL CHECKDB: ' + CONVERT(VARCHAR(10), @TotalEligible);

   -- Loop over eligible DBs and create a job per DB, scheduling day via round-robin across @StaggerDays
   DECLARE @i INT = 1;
   DECLARE @currentDB SYSNAME;

   DECLARE curSec CURSOR LOCAL FAST_FORWARD FOR SELECT DBName FROM @EligibleDBs ORDER BY RowNum;
   OPEN curSec;
   FETCH NEXT FROM curSec INTO @currentDB;
   WHILE @@FETCH_STATUS = 0
   BEGIN
      DECLARE @JobName SYSNAME = N'AG_SECONDARY_FULL_CHECKDB_' + @currentDB;
      -- Cleanup existing job if exists
      IF EXISTS (SELECT 1 FROM msdb.dbo.sysjobs WHERE name = @JobName)
      BEGIN
         PRINT 'Deleting existing job: ' + @JobName;
         EXEC msdb.dbo.sp_delete_job @job_name = @JobName;
      END

      -- Create job
      EXEC msdb.dbo.sp_add_job
         @job_name = @JobName,
```

```
        @owner_login_name = SUSER_SNAME(),
        @enabled = 1,
        @description = N'Weekly FULL DBCC CHECKDB (Ola) for ' + @currentDB + ' on readable AG secondary (SafetyModel 2)',
        @notify_level_email = 2,
        @notify_email_operator_name = @OperatorName;

    -- Job step: verify local role = SECONDARY and readable, else SKIP. Then run Ola full check.
    DECLARE @StepSQL NVARCHAR(MAX) = N'
SET NOCOUNT ON;
DECLARE @DB SYSNAME = N''' + REPLACE(@currentDB,'''','''''') + N''';
DECLARE @role_desc SYSNAME = NULL;
DECLARE @allow_conn_desc NVARCHAR(60) = NULL;
DECLARE @res NVARCHAR(MAX) = N'''';

-- get role for this DB on local instance
SELECT TOP(1) @role_desc = drs.role_desc
FROM sys.dm_hadr_database_replica_states drs
JOIN sys.availability_databases_cluster adc ON drs.group_database_id = adc.group_database_id
WHERE adc.database_name = @DB AND drs.is_local = 1;

-- get secondary read allow connections desc
SELECT TOP(1) @allow_conn_desc = ars.secondary_role_allow_connections_desc
FROM sys.dm_hadr_availability_replica_states ars
JOIN sys.availability_replicas ar ON ars.replica_id = ar.replica_id
JOIN sys.availability_groups ag ON ar.group_id = ag.group_id
JOIN sys.availability_databases_cluster adc ON ag.group_id = adc.group_id
WHERE adc.database_name = @DB AND ars.is_local = 1;

IF @role_desc IS NULL
BEGIN
  SET @res = N''SKIPPED - DB not part of AG on this instance or AG metadata not present.'';
  INSERT INTO DBA.dbo.DBA_CheckDB_Log(DatabaseName, CheckType, ResultText) VALUES (@DB, N''AG_FULL_CHECKDB'',
@res);
  RETURN;
END

IF UPPER(ISNULL(@role_desc,'')) <> ''SECONDARY''
BEGIN
  SET @res = N''SKIPPED - Local replica is not SECONDARY (role='''+ISNULL(@role_desc,''NULL'')+''')'';
  INSERT INTO DBA.dbo.DBA_CheckDB_Log(DatabaseName, CheckType, ResultText) VALUES (@DB, N''AG_FULL_CHECKDB'',
@res);
  RETURN;
END

IF UPPER(ISNULL(@allow_conn_desc,'')) NOT IN (''READ_ONLY'',''ALL'')
BEGIN
  SET @res = N''SKIPPED - Local replica does not allow read connections
(secondary_role_allow_connections_desc='''+ISNULL(@allow_conn_desc,''NULL'')+''')'';
  INSERT INTO DBA.dbo.DBA_CheckDB_Log(DatabaseName, CheckType, ResultText) VALUES (@DB, N''AG_FULL_CHECKDB'',
@res);
  RETURN;
END
```

```
-- Conditions satisfied (SafetyModel 2). Run FULL CHECKDB via Ola Hallengren
BEGIN TRY
   EXECUTE dbo.DatabaseIntegrityCheck
      @Databases = @DB,
      @CheckCommands = ''CHECKDB'',
      @PhysicalOnly = ''N'',
      @LogToTable = ''Y'';
   INSERT INTO DBA.dbo.DBA_CheckDB_Log(DatabaseName, CheckType, ResultText) VALUES (@DB, N''AG_FULL_CHECKDB'',
N''OK'');
END TRY
BEGIN CATCH
   INSERT INTO DBA.dbo.DBA_CheckDB_Log(DatabaseName, CheckType, ResultText) VALUES (@DB, N''AG_FULL_CHECKDB'',
ERROR_MESSAGE());
   THROW;
END CATCH;
';
      -- Add job step
      EXEC msdb.dbo.sp_add_jobstep
         @job_name = @JobName,
         @step_name = N'Check role/readability and run FULL CHECKDB via Ola for ' + @currentDB,
         @subsystem = N'TSQL',
         @command = @StepSQL,
         @on_success_action = 1,
         @on_fail_action = 2;
      -- Build schedule: choose day via round-robin across @StaggerDays, starting Sunday.
      DECLARE @dayIndex INT = ((@i - 1) % @StaggerDays) + 1;  -- 1..@StaggerDays
      -- Map dayIndex (1..@StaggerDays) to actual weekday bit (1=Sunday,2=Monday...)
      DECLARE @DayBit INT;
      SELECT @DayBit = DayBit FROM @DayBits WHERE DayIndex = @dayIndex;

      -- Create a unique schedule name per DB
      DECLARE @ScheduleName NVARCHAR(128) = N'Sched_Weekly_FullCheck_' + @currentDB;

      -- Clean up existing schedule with same name (if any)
      IF EXISTS (SELECT 1 FROM msdb.dbo.sysschedules WHERE name = @ScheduleName)
      BEGIN
         EXEC msdb.dbo.sp_delete_schedule @schedule_name = @ScheduleName;
      END

      -- Add schedule: weekly on chosen weekday
      EXEC msdb.dbo.sp_add_schedule
         @schedule_name = @ScheduleName,
         @freq_type = 8,          -- weekly
         @freq_interval = @DayBit,  -- weekday bitmask (Sunday=1, Monday=2, etc)
         @active_start_time = @WeeklyStartTime;

      -- Attach schedule to job and jobserver
      EXEC msdb.dbo.sp_attach_schedule @job_name = @JobName, @schedule_name = @ScheduleName;
      EXEC msdb.dbo.sp_add_jobserver @job_name = @JobName, @server_name = @@SERVERNAME;

      PRINT 'Created job: ' + @JobName + ' scheduled on day bit ' + CONVERT(VARCHAR(10), @DayBit);
```

```
        SET @i = @i + 1;

            FETCH NEXT FROM curSec INTO @currentDB;
        END

        CLOSE curSec;
        DEALLOCATE curSec;

        PRINT 'All secondary jobs created (staggered across ' + CONVERT(VARCHAR(10), @StaggerDays) + ' nights).';
    END
    ELSE
    BEGIN
        PRINT 'No eligible secondary DBs found on this instance. No secondary jobs created.';
    END

    PRINT 'Script finished. Verify jobs in SQL Server Agent. Check DBA.dbo.DBA_CheckDB_Log for results.';
    GO
```

Notes & small recommendations

- STRING_SPLIT is used for parsing DB names (SQL Server 2016+). If you run on older SQL Server, we need to provide an alternate parser.
- The script schedules staggered DB jobs over the first @StaggerDays weekdays starting from **Sunday**.
  Example with @StaggerDays = 3: DB1 → Sunday, DB2 → Monday, DB3 → Tuesday, DB4 → Sunday, etc.
- If you prefer a different start-day (e.g., Monday) or a custom mapping (Mon/Wed/Fri), we can change the mapping quickly.
- Jobs are created to **notify the Operator on failure** (SQL Agent job-level notification).
  Ensure Database Mail + SQL Agent Alert System are configured.
- The script uses dbo.DatabaseIntegrityCheck (Ola). Ensure Ola is installed on the instance where jobs run.
  The job step will throw on error to let SQL Agent mark the job step as failed and notify the operator.
- For VLDBs you still may prefer to stagger time-of-day in addition to day-of-week if the DBs are huge; we kept the same @WeeklyStartTime for all jobs but we can vary start times by DB index if you want.