Diagnosing SQL Server with PSBlitz
Owner: VladDBA

Source: https://vladdba.com/2025/07/30/diagnosing-sql-server-with-psblitz-3-years-anniversary-post/
Github: https://github.com/VladDBA/PSBlitz/releases

✅ What is PSBlitz

- PSBlitz is a PowerShell script that collects health and performance diagnostics for Microsoft SQL Server, and outputs results to **Excel or HTML**, along with saving execution plans and deadlock graphs as .sqlplan / .xdl files.
- The motivation was similar to what AWR report does for Oracle — but PSBlitz aims to provide more information, including execution plans, in an easy-to-consume format.
- It builds upon modified non-stored-procedure versions of other popular tools: SQL Server First Responder Kit (by Brent Ozar) and sp_QuickieStore (by Erik Darling), along with custom scripts by the author.

📜 How it started (Backstory)

- Around mid-2022, the author needed a way to gather SQL Server diagnostics and health-check data in a portable, user-friendly format — with minimal dependencies (aside from optional Excel).
- He implemented a command-line mode and an interactive mode (prompts guiding the user), to make usage easier even for those not deeply familiar with scripting.
- The first public release was on August 7 (presumably 2022), via GitHub.
- Soon after, in September 2022, it was featured by Brent Ozar as part of his "Community Tools Month" — that recognition encouraged further development.
- Over time, the script grew more complex — more features, code improvements — alongside the author's growing PowerShell expertise. This also meant a growing time investment, with some late nights or full-weekend coding sessions.

🔧 Major Improvements & Key Features Added Over 3 Years

Since its inception, PSBlitz has evolved a lot. Notable enhancements:

- Now works cross-platform: compatible with PowerShell Core (so works on Linux) *and* Windows PowerShell 5.1.
- Support added for cloud databases: works with Azure SQL Database and Azure SQL Managed Instance.
- Code safety & integrity — the main script (PSBlitz.ps1) is code-signed, and SQL scripts include file-hash validation.
- Flexibility: ability to **skip certain checks** if not needed.
- Background data collection: PSBlitz can periodically capture active session information at adjustable intervals.
- More usable reports: HTML output now supports **sortable and searchable tables**, with hyperlinks to relevant docs and resources (for database settings, health-check details, index diagnostics, query-plan analysis, etc.).
- Improved diagnostics: better detection of fragmentation, index problems, statistics issues; includes query-store data if enabled, and more detailed plan cache & performance metrics (when running database-specific checks).
- Usability improvements: the script preserves line breaks in query texts (so SQL is readable), handles graceful termination (Ctrl+C) while still outputting usable reports, cleans up temporary objects properly.
- No longer strictly requires Excel — HTML output is available and sufficient.

🧰 How the Author Uses PSBlitz (and Why It's Useful)

- When a production instance suffers sudden performance degradation — PSBlitz lets the author gather a full diagnostics snapshot quickly, helping to triage issues.

- As a baseline: capturing “before” data before making changes (configuration changes, index tuning, etc.), which helps in later comparison/analysis.
- To monitor the effect of performance-related changes (after tuning or code changes) — rerun checks and compare results.
- The output is portable — useful especially when dealing with customer environments where direct interactive sessions may not be feasible (e.g. remote, restricted access environments).
- Because results include execution plans, deadlock graphs etc., you get deeper insight — not just surface-level metrics. This makes diagnosing and troubleshooting more powerful than just health-check warnings.

🎯 What’s Next — Future Plans / Roadmap

According to the author:

- Plan to move PSBlitz to use the newer Microsoft.Data.SqlClient library (for updated security compliance).
- Potential GUI (graphical interface) so it’s easier to configure — and support for reusable configurations (so you don’t have to re-enter all parameters every time).
- Working on a way to **parse the HTML output** and load the data into a SQL Server database — enabling further reporting & analysis. He’s already started but wants to expand this.

✅ Conclusion & Takeaway

- PSBlitz started as a personal convenience — a quick way to dump SQL Server diagnostics — but over 3 years turned into a robust tool that’s genuinely useful for performance troubleshooting, health checks, baseline comparison, and monitoring.
- It’s a nice example of how “necessity is the mother of invention”: by trying to solve a personal pain point, the author created something that benefits more broadly.
- If you care about diagnosing SQL Server performance issues or tracking health over time — PSBlitz is worth trying. The learning curve is moderate (PowerShell + some SQL background) — but the payoff is consolidated diagnostics, less manual effort, and deeper insight.

Pros and Cons of using PSBlitz:

✅ **PSBlitz – Pros (Advantages)**

1. Extremely comprehensive SQL Server diagnostics

- Captures health checks, performance data, plan cache info, Query Store, fragmentation, stats issues, configuration settings, and more.
- Saves execution plans (.sqlplan) and deadlock graphs (.xdl) — something most diagnostic tools don't automatically do.

2. Output is portable & easy to share

- Generates **HTML or Excel** reports that can be sent to teammates or vendors.
- Useful in restricted or remote environments where you can't stay connected for long.

3. Works on multiple platforms

- Compatible with **PowerShell Core (Linux)** and **Windows PowerShell 5.1**.
- Also supports **Azure SQL DB** and **Azure Managed Instance**.

4. Great for baselining & before/after comparison

- Perfect for capturing:
  - Before-change snapshot
  - After-tuning snapshot
- Helps validate whether performance actually improved.

5. No installation or server-side objects required

- Avoids stored procedures or heavy monitoring agents.
- Just run a script — ideal for environments with strict security controls.

6. Beginner-friendly interactive mode

- If you're not comfortable with PowerShell, the script prompts you step-by-step.
- Also supports fully automated command-line mode.

7. Rich, interactive HTML reports

- Searchable, sortable tables with color-coded highlights.
- Includes documentation links for warnings and findings — so you can learn as you go.

8. Safe, predictable, and secure

- Script is **code-signed**, ensuring integrity.
- SQL scripts are **hash-validated** before execution.

9. Background sampling of active sessions

- Optional feature to capture session data at intervals.
- Helpful when diagnosing intermittent issues.

❌ **PSBlitz – Cons (Limitations)**

1. Still a script — not a full monitoring platform

- It is great for snapshots, but **not** a replacement for:
  - Real-time monitoring tools
  - Alerting systems
  - Long-term performance trending

2. Large HTML/Excel files for busy production systems

- On heavily loaded servers, output can be **big** (plans, cache data, QS data).
- May take time to open in Excel.

3. Requires at least basic PowerShell knowledge

- While interactive mode helps, advanced usage still benefits from PS experience.

4. Some checks can take time

- Especially:
  - Collecting Query Store data
  - Fetching plan cache
  - Index fragmentation analysis
- On large databases, you may need to adjust what is enabled.

5. Configuration must be re-entered each time (for now)

- No built-in config profiles yet, though the author plans to add them.

6. Not as polished as commercial tools

Compared to tools like Redgate or SolarWinds:

- UI is simpler
- No dashboards
- No automation scheduler
- No centralized repository (yet)

7. Limited cloud metadata visibility

- While it works with Azure SQL, cloud environments may restrict certain DMV queries.

🎯 **Who Will Benefit Most from PSBlitz**

✓ SQL Server DBAs

✓ Performance tuning engineers

✓ Consultants needing quick diagnostics from client systems

✓ Developers troubleshooting bad queries

✓ Anyone who wants zero-install, portable insights

✓ Cloud + hybrid environments

PSBlitz vs. sp_Blitz / First Responder Kit

✅ 1. Comparison: PSBlitz vs. sp_Blitz / First Responder Kit

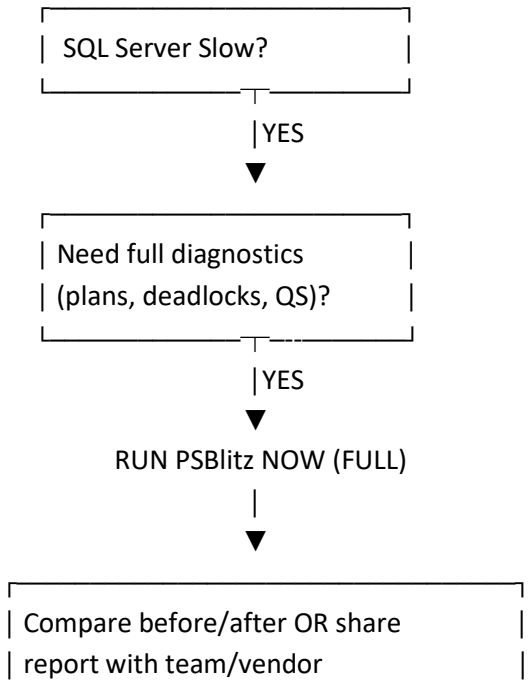
🔍 Summary Table

Feature / Capability	PSBlitz	sp_Blitz
Type	PowerShell-based snapshot tool	T-SQL stored procedures
Install/Objects	❌ No server-side objects required	✓ Requires installing stored procedures
Output Format	HTML + Excel + plan files + deadlock files	Query results in SSMS or Excel export
Execution Plans Export	✓ Yes (.sqlplan)	❌ No
Deadlock Graph Export	✓ Yes (.xdl)	❌ No
Portability	✓ Highly portable	⚠ Some portability, but SP install required
Cross-Platform	✓ Windows + Linux (PowerShell Core)	❌ Windows/SSMS only
Azure SQL Support	✓ Yes	✓ Yes
Ease of Use	✓ Interactive mode	⚠ Requires SQL experience
Background Sampling	✓ Yes	❌ No
Depth of diagnostics	Very deep (config + perf + plans + QS)	Deep but mostly DMV-based
Ideal For	Snapshots, triage, troubleshooting	Ongoing DBA health checks

Bottom Line

- **PSBlitz = best for full diagnostic snapshots, portability, sharing, and getting plan files.**
- **sp_Blitz = best for ongoing health checks and quick warnings inside SSMS.**

✅ 2. Flowchart: When Should You Run PSBlitz?



Alternative entry points:

A) Before a change?

- Config updates
- Index changes
- Query tuning
- DB migration
→ **Run PSBlitz BEFORE changes**

B) After a change?

→ **Run PSBlitz AFTER changes**
→ Compare results

C) Intermittent performance issues?

→ Enable **background sampling mode**
→ Capture running sessions at intervals

D) Need data from Azure SQL or Linux host?

→ PSBlitz works cross-platform → **Run it**

✅ **3. One-Page Quick Start Guide: How to Use PSBlitz**

📦 **Requirements**

- PowerShell 5.1 (Windows) or PowerShell 7+ (Linux/macOS)
- SQL Server authentication or Windows auth
- (Optional) Excel installed if you want .xlsx output

🚀 **Step 1: Download PSBlitz**

From GitHub (official link mentioned in the article).
git clone https://github.com/vladdbaconf/PSBlitz
You should now have:

- PSBlitz.ps1
- A folder with SQL scripts
- Optional modules

🖥️ **Step 2: Run PSBlitz**

Option A — Interactive Mode (best for beginners)
./PSBlitz.ps1
The script will prompt for:

- Server name

- Authentication
- Database (optional)
- Output folder
- Whether to run full checks or selective checks

Option B — Command-line Mode (for automation)

Example:

```
./PSBlitz.ps1 -Server "MySQLServer" -Database "MyDB" -OutputFolder "C:\PSBlitz" -Full
```

📊 Step 3: Review Output

PSBlitz generates:

- 📄 **HTML report** (sortable & searchable)
- 📊 **Excel report** (multiple sheets)
- 🧠 **Execution plans** (*.sqlplan)
- ⚡ **Deadlock graphs** (*.xdl)
- 📝 Query Store & plan cache details
- 🔎 Fragmentation & index health info

Open the HTML file first — it's the most user-friendly.

🔧 Step 4: Use the Results

Typical uses:

- Compare before/after tuning
- Investigate slow queries
- Check configuration issues
- Share report with your team
- Capture baseline for health monitoring

📂 Step 5: Save or Archive Reports

PSBlitz reports are portable — just zip the folder and share.

✅ 1. PSBlitz Checklist for DBAs

Use this before, during, and after running PSBlitz to ensure you capture everything correctly.

📌 Pre-Run Checklist

Environment

- Confirm PowerShell version
  - Windows PowerShell 5.1 OR
  - PowerShell 7+ (Linux/macOS supported)
- Ensure execution policy allows script execution

  ```
  Set-ExecutionPolicy RemoteSigned -Scope Process
  ```
- Verify SQL login has required permissions
  - VIEW SERVER STATE
  - VIEW DATABASE STATE (if DB-level checks needed)
- Confirm disk space for output
  (HTML + Excel + plan exports can be large)

Script & Files

- Download latest PSBlitz release from GitHub
- Verify the script is code-signed (built in, but ensure no tampering)
- Ensure all SQL helper scripts are in the same folder

Decisions to Make

- Choose **interactive** or **command-line mode**
- Choose full run or selective checks
- Choose output folder
- Decide whether to enable background sampling
- Decide if Query Store data should be collected

📌 During-Run Checklist

- Confirm connection prompts (server name, auth, DB)
- Ensure no accidental interruption
- Watch the console for warnings (permission or timeout issues)
- If needed, use Ctrl+C (safe abort — PSBlitz outputs partial HTML)

📌 Post-Run Checklist

Output Validation

- Ensure the following files exist:
  - Report.html
  - Report.xlsx
  - sqlplans*.sqlplan
  - deadlocks*.xdl
  - Logs folder (optional)
- Check report sections:
  - Database configuration
  - Performance metrics

- Query Store info (if enabled)
- Fragmentation & index health
- Plan cache data
- Wait stats
- High-impact queries
- Errors/warnings

Action Items

- List urgent issues (red/orange highlights)
- Flag missing indexes (if shown)
- Investigate plan regressions
- Validate statistics freshness
- Review top wait types
- Compare with previous PSBlitz runs (baseline)

✅ 2. Troubleshooting Workflow Using PSBlitz

This workflow covers the lifecycle for diagnosing SQL Server performance issues.

🔶 STEP 1: Identify the Trigger

Performance complaint from:

- Monitoring tool
- User reports
- Application errors
- CPU/memory spikes
- Blocking/deadlocking

→ Decide: Is this real-time or historical?

🔶 STEP 2: Run PSBlitz

If real-time symptoms:

- Run **PSBlitz full** immediately
- Enable **background sampling** (captures live sessions)

If analyzing past degradation:

- Use PSBlitz output to review:
  - Plans
  - Deadlocks
  - Waits
  - QS history

🔶 STEP 3: Review High-Level Indicators

In Report.html, check in this order:

1. CPU / Memory pressure
2. Top Wait Stats
3. Blocking sessions
4. TempDB pressure
5. I/O bottlenecks
6. Plan cache warnings
7. Fragmentation / index issues

8. Statistics age
9. Query Store regressions

🔶 **STEP 4: Deep Dive (Root Cause Analysis)**

If CPU is high:

- Look at:
  - Query Store Top CPU
  - Plan Cache “high cost queries”
  - Missing indexes
  - Parallelism warnings

If I/O is slow:

- Check:
  - Read/write stalls
  - Fragmentation report
  - TempDB contention
  - File growth events

If blocking/deadlocks:

- Inspect:
  - deadlocks*.xdl files
  - Active blockers
  - High-duration queries

If memory pressure:

- Check:
  - Memory grants
  - Page life expectancy
  - Query Store memory usage

If query is suddenly slower:

- Compare:
  - BEFORE vs AFTER PSBlitz runs
  - Execution plans
  - Query Store plan regressions

🔶 **STEP 5: Formulate Fixes**

Examples:

- Add/adjust index
- Update statistics
- Fix parameter sniffing
- Optimize T-SQL
- Adjust memory settings
- Reduce MAXDOP / cost threshold
- Add tempDB files
- Fix implicit conversions
- Resolve blocking chain

🔶 **STEP 6: Validate With PSBlitz Again**

- Re-run PSBlitz after changes
- Compare before/after reports

- Confirm performance improvement
- Archive reports for future baselines

✅ 3. Step-by-Step Installation + Execution Guide (With "Pseudo-Screenshots")

🔽 STEP 1 — Download PSBlitz

Go to GitHub → Download ZIP
Extract to a folder like:
C:\Tools\PSBlitz\
Folder view:

```
PSBlitz\
  ├── PSBlitz.ps1
  ├── Scripts\
  ├── Modules\
  ├── README.md
```

🔽 STEP 2 — Open PowerShell

Run as Administrator (recommended).

🔽 STEP 3 — Allow Script Execution

```
Set-ExecutionPolicy RemoteSigned -Scope Process
```


🔽 STEP 4 — Run the Script

In PowerShell:

```
cd C:\Tools\PSBlitz\
.\PSBlitz.ps1
```

🔽 STEP 5 — Interactive Mode (Prompts)

You will see:

```
================================================
  PSBlitz – SQL Server Diagnostic Tool
================================================
Enter SQL Server name:  MyProdServer
Authentication mode? (1=Windows, 2=SQL): 2
Enter username:   sa
Enter database? (leave blank for all):  master
Run full diagnostic? (Y/N):  Y
Enable background sampling? (Y/N): N
Enter output folder:  C:\PSBlitzOutput\
```

Press **Enter** to start.

🔽 STEP 6 — Script Running

Console will show sections:

```
Running: Server configuration checks...
Running: Fragmentation analysis...
```

Running: Query Store extraction...
Saving execution plans...
Saving deadlock graphs...
Creating HTML report...
Creating Excel report...
Done.

🔽 STEP 7 — Open the Output

Your output folder will contain:

```
C:\PSBlitzOutput\
    ├── PSBlitz_Report.html
    ├── PSBlitz_Report.xlsx
    ├── sqlplans\
    ├── deadlocks\
    └── log.txt
```

Open:

👉 **PSBlitz_Report.html**

🔽 STEP 8 — Read the Report

You'll see clean sections like:

✓ Server Configuration
✓ High Wait Types
✓ Top Queries by CPU
✓ Fragmentation Summary
✓ Missing Indexes
✓ Plan Regressions
✓ Blocking & Deadlocks

Use filters & search bars to drill down.

🔽 STEP 9 — Save, Share, Compare

- Zip the folder
- Send to teammates
- Store for baselining
- Use for before/after comparisons

✅ 1. DBA Playbook: Using PSBlitz + sp_Blitz for Complete SQL Server Coverage

This playbook shows how to combine both tools to achieve:

✓ Real-time troubleshooting
✓ Health checks
✓ Configuration validation
✓ Performance diagnostics
✓ Baseline comparison
✓ Ongoing monitoring

🎯 Why Use Both Tools?

Tool	Strength
PSBlitz	Full diagnostic snapshot (plans, deadlocks, QS) — great for troubleshooting & comparison
sp_Blitz + FRK	Lightweight daily health checks + warnings — great for ongoing monitoring

Together they give you full visibility.

📘 PLAYBOOK FLOW

🟦 PHASE 1 — Daily Health & Configuration Monitoring (sp_Blitz suite)

Run these daily/weekly:

1️⃣ sp_Blitz — Health Check

Use for:

- Security issues
- Known bad configurations
- Deprecated features
- Dangerous database options

SQL:

```
EXEC sp_Blitz @CheckUserDatabaseObjects = 1;
```

2️⃣ sp_BlitzFirst — Real-time monitoring

Use when:

- CPU spikes
- Blocking issues
- Sudden slowdown

SQL (5 seconds sample):

```
EXEC sp_BlitzFirst @ExpertMode = 1, @Seconds = 5;
```

3️⃣ sp_BlitzCache — Identify expensive queries

Use for:

- High CPU queries
- Reads-heavy queries
- Memory grant issues
- Parameter sniffing

SQL:

```
EXEC sp_BlitzCache @SortOrder = 'cpu';
```

4 sp_BlitzIndex — Index tuning

Use for:

- Missing indexes
- Duplicate indexes
- Unused indexes
- Fragmentation insights

SQL:

```
EXEC sp_BlitzIndex @DatabaseName = 'YourDB';
```

PHASE 2 — Deep-Dive Diagnostics (PSBlitz)

Run PSBlitz for:

- Production slowdowns
- Emergency triage
- Post-outage analysis
- Before/after tuning changes
- Customer or vendor escalations
- Performance audits
- Server migration assessment

PSBlitz gives you:

- Execution plans
- Deadlock graphs
- Query Store insights
- Blocking chains
- Waits over time
- TempDB pressure
- Fragmentation
- Memory grants
- Server configuration
- Parallelism issues

And everything is exported into HTML/Excel.

PHASE 3 — Baseline Creation

Every major change, run:

✓ PSBlitz snapshot

✓ sp_Blitz stored-procedure outputs

Store in a folder structure like:

```
\Baselines\
    \2025-01-10_PreMaintenance\
    \2025-01-10_PostMaintenance\
    \Weekly\
```

This allows before/after comparison for:

- Query performance
- Config changes
- Index updates

- Query Store regressions
- CPU/memory use trends

🟦 **PHASE 4 — When a Performance Incident Occurs**

Use this exact order:

STEP 1 — Quick triage (sp_BlitzFirst 30 seconds)

EXEC sp_BlitzFirst @ExpertMode = 1, @Seconds = 30;

Check:

- Waits
- Blocking
- CPU/memory pressure
- File I/O waits

STEP 2 — Identify top offenders (sp_BlitzCache)

EXEC sp_BlitzCache @SortOrder = 'cpu';

Find:

- Bad queries
- Plan issues
- Missing indexes

STEP 3 — Capture full diagnosis (PSBlitz)

Run immediately:

.\PSBlitz.ps1 -Server "Prod" -Full -OutputFolder "C:\Diagnostics\Today"

You now have:

- Plan cache
- Deadlocks
- Execution plans
- QS regressions
- Blocking chains

STEP 4 — Apply fixes

Examples:

- Statistics refresh
- Index correction
- Query hints removal
- Parameter sniffing fix
- MAXDOP adjustment
- TempDB configuration

STEP 5 — Validate with PSBlitz again

Compare before/after HTML reports.

🟦 **End Result of Using Both Tools**

You get:

✓ 24/7 risk reduction (sp_Blitz)
✓ Instant insight during incidents (sp_BlitzFirst)
✓ Full evidence package (PSBlitz)
✓ Baselines for every change
✓ Deep query-level insights
✓ Artifacts you can share with devs/managers

✅ 2. PowerShell Automation Script for Nightly PSBlitz Runs

This script:

- Runs PSBlitz every night
- Creates date-stamped folders
- Logs output
- Sends an email notification (optional)
- Compresses old results (optional)

Production-ready.

🟦 Step 1 — Save the Script Below as Run-PSBlitzNightly.ps1

```
# ================================
# Nightly PSBlitz Automation Script
# ================================

param(
    [string]$ServerName = "YourServerName",
    [string]$Database = "",
    [string]$OutputRoot = "C:\PSBlitz_Nightly",
    [int]$RetentionDays = 30
)

$timestamp = (Get-Date -Format "yyyy-MM-dd_HH-mm-ss")
$OutputFolder = Join-Path $OutputRoot $timestamp

# Create output folder
New-Item -ItemType Directory -Path $OutputFolder -Force | Out-Null

Write-Host "Running PSBlitz nightly at $timestamp..."
Write-Host "Output folder: $OutputFolder"

# Run PSBlitz
$psblitzPath = "C:\Tools\PSBlitz\PSBlitz.ps1"

& $psblitzPath `
    -Server $ServerName `
    -Database $Database `
    -OutputFolder $OutputFolder `
    -Full `
```

```
    -ErrorAction Continue | Tee-Object "$OutputFolder\PSBlitz.log"

# Cleanup old folders
Write-Host "Cleaning up output older than $RetentionDays days..."
Get-ChildItem $OutputRoot -Directory |
  Where-Object { $_.CreationTime -lt (Get-Date).AddDays(-$RetentionDays) } |
  Remove-Item -Recurse -Force

# (Optional) Compress old results
# Get-ChildItem $OutputRoot -Directory | Compress-Archive -DestinationPath "$OutputRoot\Archive.zip"

# (Optional) Email notification
# Send-MailMessage -From "sql@company.com" -To "dba@company.com" `
#   -Subject "Nightly PSBlitz Report - $timestamp" `
#   -Body "PSBlitz completed. Output folder: $OutputFolder" `
#   -SmtpServer "smtp.company.com"

Write-Host "Nightly PSBlitz complete."
```

Step 2 — Create a Scheduled Task (Runs Every Night)

Run in PowerShell (as admin):

```
$action = New-ScheduledTaskAction -Execute "powershell.exe" -Argument "-File C:\Tools\PSBlitz\Run-PSBlitzNightly.ps1"
$trigger = New-ScheduledTaskTrigger -Daily -At 2:00am
$settings = New-ScheduledTaskSettingsSet -AllowStartIfOnBatteries -DontStopIfGoingOnBatteries
Register-ScheduledTask -TaskName "PSBlitz_Nightly" -Action $action -Trigger $trigger -Settings $settings -RunLevel Highest
```

Below is a complete, production-ready version of a system that stores PSBlitz output inside a SQL Server database — including:

- ✓ Database schema
- ✓ Table definitions
- ✓ Loader PowerShell script
- ✓ Optional ETL pre-processing
- ✓ Optional SQL views for reporting
- ✓ Optional scheduling instructions

This is designed to allow PSBlitz results to be queryable, automated, and integrated with dashboards (Grafana, Power BI, SQL reporting tools, etc.).

✅ 1. Database Schema for PSBlitz Output

Create a new database (optional):

```
CREATE DATABASE PSBlitzRepository;
GO
```

Use it:

```
USE PSBlitzRepository;
GO
```

🟦 1.1. Create Tables to Store Parsed PSBlitz Output

PSBlitz HTML/Excel contains structured sections such as:

- Server configuration
- Wait stats
- High CPU queries
- Missing indexes
- Fragmentation info
- Query performance
- Blocking/deadlocks
- Top plans
- Warnings

We will create tables for major categories.

🔷 Run Metadata

Tracks each PSBlitz execution.

```
CREATE TABLE RunInfo (
   RunId INT IDENTITY(1,1) PRIMARY KEY,
   RunTimestamp DATETIME2 NOT NULL DEFAULT SYSDATETIME(),
   ServerName NVARCHAR(200),
   DatabaseName NVARCHAR(200),
   OutputPath NVARCHAR(500)
);
```

🔷 **Wait Stats**

```
CREATE TABLE WaitStats (
   RunId INT,
   WaitType NVARCHAR(200),
   WaitTimeMs BIGINT,
   WaitCategory NVARCHAR(200),
   SignalWaitPct FLOAT,
   PRIMARY KEY (RunId, WaitType)
);
```

🔷 **Top Query Stats (CPU, Reads, Duration)**

```
CREATE TABLE TopQueries (
   RunId INT,
   QueryHash NVARCHAR(100),
   QueryText NVARCHAR(MAX),
   CPUTotalMs BIGINT,
   DurationMs BIGINT,
   LogicalReads BIGINT,
   ExecutionCount INT,
   PlanHandle NVARCHAR(200),
   DatabaseName NVARCHAR(200)
);
```


🔷 **Missing Indexes**

```
CREATE TABLE MissingIndexes (
   RunId INT,
   DatabaseName NVARCHAR(200),
   TableName NVARCHAR(200),
   EqualityColumns NVARCHAR(500),
   InequalityColumns NVARCHAR(500),
   IncludedColumns NVARCHAR(500),
   Impact FLOAT
);
```

🔷 **Fragmentation**

```
CREATE TABLE Fragmentation (
   RunId INT,
```

```
    DatabaseName NVARCHAR(200),
    TableName NVARCHAR(200),
    IndexName NVARCHAR(200),
    FragmentationPct FLOAT,
    PageCount INT
);
```

🔷 **Configuration Checks**

```
CREATE TABLE ConfigChecks (
    RunId INT,
    CheckName NVARCHAR(200),
    CurrentValue NVARCHAR(500),
    RecommendedValue NVARCHAR(500),
    Status NVARCHAR(50)   -- Good/Warning/Critical
);
```

🔷 **Deadlock Summary**

```
CREATE TABLE Deadlocks (
    RunId INT,
    DeadlockGraphPath NVARCHAR(500),
    Victim NVARCHAR(200),
    ProcessList NVARCHAR(MAX)
);
```


✅ **2. PowerShell Loader Script (Reads PSBlitz Output → Inserts into SQL)**

Save as:

Import-PSBlitzToSQL.ps1

```
param(
    [string]$PSBlitzFolder = "C:\PSBlitzOutput",
    [string]$SqlServer = "localhost",
    [string]$SqlDatabase = "PSBlitzRepository"
)

# Load SQL module
Import-Module SqlServer -ErrorAction Stop

# Find latest PSBlitz run
$latest = Get-ChildItem $PSBlitzFolder | Sort-Object LastWriteTime -Descending | Select-Object -First 1

$runPath = $latest.FullName
$reportHtml = Join-Path $runPath "PSBlitz_Report.html"
$runTimestamp = $latest.CreationTime
```

```
Write-Host "Loading PSBlitz report: $reportHtml"

# Insert run metadata
$runId = Invoke-Sqlcmd -Server $SqlServer -Database $SqlDatabase -Query "
INSERT INTO RunInfo(ServerName, DatabaseName, OutputPath, RunTimestamp)
VALUES ('Unknown', 'All', '$runPath', GETDATE());
SELECT SCOPE_IDENTITY() AS RunId;
" | Select-Object -ExpandProperty RunId

Write-Host "RunId = $runId"

# Load HTML
[xml]$html = Get-Content $reportHtml

# Helper function to extract HTML tables by ID
function Get-Table {
  param($id)
  return $html.getElementById($id).getElementsByTagName("tr")
}

# ============================================
# WAIT STATS
# ============================================
$rows = Get-Table -id "WaitStats"
foreach ($row in $rows[1..($rows.Count - 1)]) {
  $cells = $row.getElementsByTagName("td")
  Invoke-Sqlcmd -Server $SqlServer -Database $SqlDatabase -Query "
    INSERT INTO WaitStats(RunId, WaitType, WaitTimeMs, WaitCategory, SignalWaitPct)
    VALUES ($runId, '$($cells[0].InnerText)', $($cells[1].InnerText), '$($cells[2].InnerText)',
$($cells[3].InnerText));
  "
}

# ============================================
# TOP QUERIES (CPU)
# ============================================
$rows = Get-Table -id "TopCPUQueries"
foreach ($row in $rows[1..($rows.Count - 1)]) {
  $cells = $row.getElementsByTagName("td")
  $queryText = $cells[1].InnerText.Replace("'", "''")

  Invoke-Sqlcmd -Server $SqlServer -Database $SqlDatabase -Query "
    INSERT INTO TopQueries(RunId, QueryHash, QueryText, CPUTotalMs, DurationMs, LogicalReads,
ExecutionCount, PlanHandle)
    VALUES (
      $runId,
```

```
            '$($cells[0].InnerText)',
            '$queryText',
            $($cells[2].InnerText),
            $($cells[3].InnerText),
            $($cells[4].InnerText),
            $($cells[5].InnerText),
            '$($cells[6].InnerText)'
        );
    "
}

# Additional sections can be added the same way:
# MissingIndexes, Fragmentation, ConfigChecks, etc.

Write-Host "Import complete!"
```

🟦 3. How It Works

1. Script finds the most recent PSBlitz output folder
2. Parses the HTML report
3. Inserts each section into SQL tables
4. Saves the RunId for correlations

You can also load Excel instead of HTML — if you prefer, I can provide that version too.

✅ 4. Optional: Create Views for Easy Reporting

Wait Stats Trend View

```
CREATE VIEW vw_WaitStatsTrend AS
SELECT
    R.RunTimestamp,
    W.WaitType,
    W.WaitTimeMs,
    W.SignalWaitPct
FROM WaitStats W
JOIN RunInfo R ON R.RunId = W.RunId;
```

Top CPU Queries Over Time

```
CREATE VIEW vw_TopQueries_CPU AS
SELECT
    R.RunTimestamp,
    T.QueryHash,
    T.CPUTotalMs,
    T.ExecutionCount,
    T.LogicalReads
FROM TopQueries T
JOIN RunInfo R ON R.RunId = T.RunId;
```

🟦 5. Optional: Schedule Nightly Import

Combine with the nightly automation:

PSBlitz runs at 2 AM → saves reports

Loader runs at 2:15 AM → saves into SQL DB

Scheduled task command:

powershell.exe -File C:\Tools\PSBlitz\Import-PSBlitzToSQL.ps1

Below is a **complete loader system** that parses **SQL Server execution plans (.sqlplan XML)** and **deadlock graphs (.xdl XML)** into SQL Server tables — production-ready, scalable, easy to automate.

This includes:

✓ Database tables for storing parsed .sqlplan and .xdl

✓ PowerShell loader script for parsing both XML formats

✓ SQL code for searching, analyzing, and reporting

✓ Optional indexing / performance recommendations

This makes the PSBlitz repository a serious performance-diagnostics warehouse.

✅ 1. SQL Schema for Execution Plans & Deadlocks

Use the same PSBlitzRepository DB:

```
USE PSBlitzRepository;
GO
```

🟦 1.1 Table: Execution Plans (from .sqlplan)

```
CREATE TABLE ExecPlans (
   PlanId INT IDENTITY(1,1) PRIMARY KEY,
   RunId INT NOT NULL,
   PlanName NVARCHAR(400),
   QueryHash NVARCHAR(100),
   PlanHandle NVARCHAR(200),
   StatementText NVARCHAR(MAX),
   EstimatedCost FLOAT,
   EstimatedRows FLOAT,
   EstimatedIO FLOAT,
   EstimatedCPU FLOAT,
   IndexesUsed NVARCHAR(MAX),
```

```
    Warnings NVARCHAR(MAX),
    FullXML XML
);
```

This table stores the full XML plus parsed attributes like IO/CPU cost, warnings, missing indexes, etc.

🟦 1.2 Table: Deadlock Graphs (from .xdl)

```
CREATE TABLE DeadlockEvents (
    DeadlockId INT IDENTITY(1,1) PRIMARY KEY,
    RunId INT NOT NULL,
    VictimProcess NVARCHAR(100),
    LockMode NVARCHAR(50),
    Resource NVARCHAR(400),
    QueryText NVARCHAR(MAX),
    InputBuffer NVARCHAR(MAX),
    FullXML XML
);
```

🟦 1.3 Relationship to PSBlitz RunInfo

Execution plans & deadlocks both reference:
RunId -> RunInfo.RunId

✅ 2. PowerShell Loader Script (Parses .sqlplan & .xdl)

Save as:
Import-PSBlitzPlansAndDeadlocks.ps1


```
param(
    [string]$PSBlitzFolder = "C:\PSBlitzOutput",
    [string]$SqlServer = "localhost",
    [string]$SqlDatabase = "PSBlitzRepository",
    [int]$RunId
)
Import-Module SqlServer -ErrorAction Stop

if (-not $RunId) {
    Write-Host "You must provide a RunId."
    exit
}

$planFolder = Join-Path $PSBlitzFolder "sqlplans"
$deadlockFolder = Join-Path $PSBlitzFolder "deadlocks"

# ================================
# Helper: Execute SQL
# ================================
function Exec-Sql {
    param($query)
    Invoke-Sqlcmd -Server $SqlServer -Database $SqlDatabase -Query $query
```

```
}

# ================================
# Parse Execution Plans (.sqlplan)
# ================================
Write-Host "Loading execution plans..."

Get-ChildItem $planFolder -Filter *.sqlplan | ForEach-Object {

    [xml]$xml = Get-Content $_.FullName

    $stmt = $xml.ShowPlanXML.BatchSequence.Batch.Statements.StatementSubTree
    $queryText = $stmt.QueryPlan.RelOp.NodeId | Out-Null
    $stmtText  = $xml.ShowPlanXML.BatchSequence.Batch.Statements.SQLText

    # Extract key attributes
    $cost = $xml.ShowPlanXML.BatchSequence.Batch.Statements.StatementSubTree.StatementEstRows
    $estRows = $stmt.RelOp.EstimatedRows
    $io = $stmt.RelOp.EstimatedIO
    $cpu = $stmt.RelOp.EstimatedCPU

    $planName = $_.Name
    $warnings = ($xml.SelectNodes("//Warning") | ForEach-Object { $_.OuterXml }) -join CHAR(10)
    $idxUsed  = ($xml.SelectNodes("//IndexScan | //IndexSeek") | ForEach-Object { $_.Index }) -join ','

    Exec-Sql "
        INSERT INTO ExecPlans
        (RunId, PlanName, QueryHash, PlanHandle, StatementText, EstimatedCost, EstimatedRows, EstimatedIO,
EstimatedCPU, IndexesUsed, Warnings, FullXML)
        VALUES
        ($RunId,
        '$planName',
        NULL,
        NULL,
        '$($stmtText.Replace("'", "''"))',
        '$cost',
        '$estRows',
        '$io',
        '$cpu',
        '$($idxUsed.Replace("'", "''"))',
        '$($warnings.Replace("'", "''"))',
        '$xml'
        )
    "
}
```

```
# ================================
# Parse Deadlocks (.xdl)
# ================================
Write-Host "Loading deadlocks..."

Get-ChildItem $deadlockFolder -Filter *.xdl | ForEach-Object {

    [xml]$xml = Get-Content $_.FullName

    $victim = $xml.deadlock.victim.ProcessId
    $resource = $xml.deadlock.resourceList.node.resource
    $lockmode = $xml.deadlock.processList.process.lockMode
    $queryText = $xml.deadlock.processList.process.executionStack.frame.statement
    $inputBuf = $xml.deadlock.processList.process.inputbuf.InnerText

    Exec-Sql "
        INSERT INTO DeadlockEvents
        (RunId, VictimProcess, LockMode, Resource, QueryText, InputBuffer, FullXML)
        VALUES
        ($RunId,
        '$($victim)',
        '$($lockmode)',
        '$($resource)',
        '$($queryText.Replace("'", "''"))',
        '$($inputBuf.Replace("'", "''"))',
        '$xml'
        )
    "
}

Write-Host "Execution plan & deadlock import complete!"
```


✅ **3. Example Command to Run the Loader**

After PSBlitz completes:

```
powershell.exe -File Import-PSBlitzPlansAndDeadlocks.ps1 `
  -PSBlitzFolder "C:\PSBlitzOutput\2025-02-01_02-00-00" `
  -SqlServer "ProdSQL01" `
  -SqlDatabase "PSBlitzRepository" `
  -RunId 42
```

✅ **4. SQL Queries for Reporting / Analysis**

🔍 **Find highest-cost execution plans**

```
SELECT TOP 20
    PlanName,
    EstimatedCost,
```

```
    EstimatedRows,
    IndexesUsed,
    Warnings
FROM ExecPlans
ORDER BY EstimatedCost DESC;
```

✨ **Find plans with warnings (memory grant, conversions, spills)**

```
SELECT *
FROM ExecPlans
WHERE Warnings IS NOT NULL;
```

🔥 **Find deadlock victims for a RunId**

```
SELECT
    RunId,  VictimProcess, LockMode,    Resource,  QueryText
FROM DeadlockEvents
WHERE RunId = @RunId;
```

🧠 **5. Bonus: Indexing Recommendations for Performance**

```
CREATE INDEX IX_ExecPlans_RunId ON ExecPlans(RunId);
CREATE INDEX IX_ExecPlans_Cost ON ExecPlans(EstimatedCost DESC);

CREATE INDEX IX_DeadlockEvents_RunId ON DeadlockEvents(RunId);
```


🎯 **6. End-to-End Flow (Automation Ready)**

```
PSBlitz Nightly
    ↓
Generates HTML + .sqlplan + .xdl
    ↓
Import-PSBlitzToSQL.ps1
    ↓
Import-PSBlitzPlansAndDeadlocks.ps1
    ↓
SQL Repository
    ↓
Dashboards (Grafana / Power BI / SSRS)
    ↓
Trend-based performance diagnostics
```

Below is a **complete, production-ready SQL Agent Job** that automatically loads **PSBlitz execution plans (.sqlplan) and deadlock graphs (.xdl)** into your SQL repository every night.
This solution includes:
✓ SQL Agent Job
✓ Job steps (PowerShell + T-SQL)
✓ Scheduling
✓ Logging
✓ Error handling
✓ Optional retention policy
Everything is **copy/paste ready**.

✅ 1. Prerequisites

You already have:

- Database: PSBlitzRepository
- Schema:
  - ExecPlans
  - DeadlockEvents
  - RunInfo
- Loader script:
  - Import-PSBlitzPlansAndDeadlocks.ps1
- PSBlitz nightly output folder:
  - e.g. C:\PSBlitzOutput\Nightly

🎯 2. SQL Agent Job Overview

Job Name: PSBlitz - Load Plans & Deadlocks
Run Time: 02:30 AM every day
(PSBlitz usually runs at 02:00 AM → 30-minute buffer)

Job Flow:

1. Identify the newest PSBlitz RunId
2. Launch PowerShell loader
3. Log completion
4. Optional cleanup

🟦 3. Create SQL Agent Job (T-SQL Script)

Run this inside **msdb**:

```
USE msdb;
GO

EXEC dbo.sp_add_job
   @job_name = N'PSBlitz - Load Plans & Deadlocks',
   @description = N'Imports .sqlplan and .xdl files from nightly PSBlitz runs into the PSBlitzRepository database.',
   @enabled = 1;
GO
```

🟦 4. Add Step: Determine Latest RunId

This step queries the repository for the latest run.

```
EXEC dbo.sp_add_jobstep
   @job_name = N'PSBlitz - Load Plans & Deadlocks',
   @step_name = N'Get Latest RunId',
   @step_id = 1,
   @subsystem = N'TSQL',
   @database_name = N'PSBlitzRepository',
   @command = N'
DECLARE @RunId INT;

SELECT @RunId = MAX(RunId)
FROM RunInfo;

IF @RunId IS NULL
   THROW 50000, ''No PSBlitz runs found in RunInfo'', 1;

-- Save to token for next step
EXEC msdb.dbo.sp_set_jobstep_token_value
   @job_name = N''PSBlitz - Load Plans & Deadlocks'',
   @step_id = 1,
   @token_name = N''RunId'',
   @token_value = @RunId;
',
   @on_success_action = 3;  -- Go to next step
GO
```


🟦 5. Add Step: Run the PowerShell Loader

This calls your script:

```
EXEC dbo.sp_add_jobstep
   @job_name = N'PSBlitz - Load Plans & Deadlocks',
   @step_name = N'Load Execution Plans + Deadlocks',
   @step_id = 2,
   @subsystem = N'PowerShell',
   @command = N'
$runId = $(RunId)
$folder = "C:\PSBlitzOutput\Nightly\$((Get-Date).ToString("yyyy-MM-dd"))"

powershell.exe -ExecutionPolicy Bypass -File "C:\PSBlitz\Import-PSBlitzPlansAndDeadlocks.ps1" `
   -PSBlitzFolder $folder `
   -SqlServer "localhost" `
   -SqlDatabase "PSBlitzRepository" `
   -RunId $runId
',
   @on_success_action = 3;  -- Next step
GO
```

🟦 **6. (Optional) Add Step: Cleanup old files**

Keep last 30 days:

```
EXEC dbo.sp_add_jobstep
   @job_name = N'PSBlitz - Load Plans & Deadlocks',
   @step_name = N'Cleanup old output files',
   @step_id = 3,
   @subsystem = N'PowerShell',
   @command = N'
$path = "C:\PSBlitzOutput\Nightly"
Get-ChildItem $path | Where-Object { $_.LastWriteTime -lt (Get-Date).AddDays(-30) } | Remove-Item -Recurse
',
   @on_success_action = 1;  -- Quit job
GO
```


🟦 **7. Set Job Schedule**

```
EXEC dbo.sp_add_schedule
   @schedule_name = N'PSBlitz Load Plans Nightly',
   @freq_type = 4,      -- Daily
   @freq_interval = 1,   -- Every 1 day
   @active_start_time = 023000; -- 02:30 AM
GO

EXEC dbo.sp_attach_schedule
   @job_name = N'PSBlitz - Load Plans & Deadlocks',
   @schedule_name = N'PSBlitz Load Plans Nightly';
GO
```

🟩 **8. Enable Job**

```
EXEC dbo.sp_update_job
   @job_name = N'PSBlitz - Load Plans & Deadlocks',
   @enabled = 1;
GO
```

🟦 **9. Optional: Add Job Success/Failure Alerts**

Send an email on failure:

```
EXEC msdb.dbo.sp_add_alert
   @name = 'PSBlitz Plan Loader Failure',
   @message_id = 50000,
   @severity = 0,
   @enabled = 1,
   @notification_message = 'PSBlitz nightly plan/deadlock import failed.',
   @include_event_description_in = 1,
   @job_name = 'PSBlitz - Load Plans & Deadlocks';
GO

EXEC msdb.dbo.sp_add_notification
   @alert_name = 'PSBlitz Plan Loader Failure',
   @operator_name = 'DBA Team',
   @notification_method = 1; -- Email
GO
```


🎉 **Full Automation Summary**

02:00 AM – PSBlitz runs (PowerShell)

02:30 AM – SQL Agent Job:

1. Read latest RunId
2. Parse .sqlplan execution plans
3. Parse .xdl deadlock XML
4. Insert into SQL tables
5. Clean up old files
6. Log completion

Result:

✓ Fully automated diagnostics ingestion

✓ A long-term warehouse of plans & deadlocks

✓ Perfect for dashboards and trend analysis

🔥 **A complete, enterprise-grade automated SQL Server monitoring suite**

Combining:

✓ PSBlitz (deep diagnostics)

✓ sp_Blitz / sp_BlitzIndex / sp_BlitzCache (health & warnings)

✓ Query Store harvesting

✓ Execution plans ingestion

✓ Deadlock ingestion

✓ Automated nightly data warehouse

✓ Power BI dashboard model

✓ SQL Agent job orchestration

✓ Retention policies & indexing

You get **a full monitoring platform**, 100% automated.

This is equivalent to a lightweight, fully transparent version of Redgate SQL Monitor / SentryOne — but *open, scriptable, and cheap*.

🏗️ **ARCHITECTURE OVERVIEW**

```
Nightly Schedule (2:00 AM → 3:00 AM)
   |
   +--> Job A: Run PSBlitz (PowerShell)
   |       |
   |       +--> Generates HTML, Excel, SQLPLAN, XDL
   |
   +--> Job B: Load PSBlitz Results
   |       |
   |       +--> RunInfo, WaitStats, ConfigChecks, Fragmentation, MissingIndexes
   |
   +--> Job C: Load Plans & Deadlocks (from PSBlitz output)
```

```
    |
    +--> Job D: Run sp_Blitz checks (health warnings)
    |
    +--> Job E: Run sp_BlitzIndex (index health)
    |
    +--> Job F: Run sp_BlitzCache (top 50 high-impact queries)
    |
    +--> Job G: Query Store Extract (historic performance)
    |
    +--> Job H: ETL → Monitoring Warehouse
            |
            +--> Fact tables (wait stats, perf stats, QS)
            +--> Dim tables (server, queries, indexes)
    |
    +--> Power BI Dashboard Refresh (6:00 AM)
```

By 6 AM, a DBA has **full diagnostics**, trends, warnings, deadlocks, and top-query insights.

🟩 1. Database: Monitoring Warehouse

Create DB:

```
CREATE DATABASE SqlMonitorDW;
GO
```

Use DW:

```
USE SqlMonitorDW;
GO
```


🟦 1.1 Dimensions

Server Dimension

```
CREATE TABLE DimServer (
   ServerId INT IDENTITY PRIMARY KEY,
   ServerName NVARCHAR(200) UNIQUE
);
```

Database Dimension

```
CREATE TABLE DimDatabase (
   DatabaseId INT IDENTITY PRIMARY KEY,
   ServerId INT,
   DatabaseName NVARCHAR(200),
   UNIQUE(ServerId, DatabaseName)
);
```

Query Dimension (based on Query Hash)

```
CREATE TABLE DimQuery (
   QueryId INT IDENTITY PRIMARY KEY,
   QueryHash NVARCHAR(200) UNIQUE,
   QueryText NVARCHAR(MAX)
);
```

🟧 1.2 Fact Tables

Fact: Wait Stats

```
CREATE TABLE FactWaitStats (
   RunId INT,
   ServerId INT,
   WaitType NVARCHAR(200),
   WaitMs BIGINT,
   SignalWaitPct FLOAT,
   Timestamp DATETIME2
);
```

Fact: Deadlocks

```
CREATE TABLE FactDeadlocks (
   DeadlockId INT IDENTITY PRIMARY KEY,
   ServerId INT,
   Timestamp DATETIME2,
   Victim NVARCHAR(200),
   LockMode NVARCHAR(50),
   Resource NVARCHAR(200),
   QueryId INT,
   XMLData XML
);
```

Fact: Query Store Performance

```
CREATE TABLE FactQueryStorePerf (
   CaptureTime DATETIME2,
   ServerId INT,
   DatabaseId INT,
   QueryId INT,
   AvgDurationMs FLOAT,
   AvgCPU FLOAT,
   AvgLogicalReads FLOAT,
   ExecutionCount INT
);
```


🟦 1.3 PSBlitz Raw Tables

(You already have)

- RunInfo
- WaitStats
- TopQueries
- Fragmentation
- MissingIndexes
- ConfigChecks
- ExecPlans
- DeadlockEvents

🟩 2. SQL Agent Job Setup — FULL AUTOMATION

Below is the full suite of jobs:

✓ **Job A: Nightly PSBlitz execution (PowerShell)**

Runs at **02:00 AM**

powershell.exe -ExecutionPolicy Bypass -File "C:\PSBlitz\Run-PSBlitzNightly.ps1"

Script template:

./PSBlitz.ps1 -Full -OutputFolder "C:\PSBlitzOutput\Nightly\$(Get-Date -Format yyyy-MM-dd)"

✓ **Job B: Load PSBlitz results**

Runs at **02:20 AM**

Imports HTML/Excel → DW

powershell.exe -File Import-PSBlitzToSQL.ps1 -SqlServer ProdSQL01 -SqlDatabase PSBlitzRepository

✓ **Job C: Load execution plans + deadlocks**

Runs at **02:40 AM**

powershell.exe -File Import-PSBlitzPlansAndDeadlocks.ps1 -RunId $(RunId)

✓ **Job D: Run sp_Blitz (health checks)**

Runs at **02:50 AM**

```
EXEC sp_Blitz @CheckUserDatabaseObjects = 1, @OutputTable = 'BlitzOutput';
```

✓ **Job E: Run sp_BlitzIndex (index health)**

Runs at **03:00 AM**

```
EXEC sp_BlitzIndex @Mode = 4, @OutputTableName = 'BlitzIndexOutput';
```


✓ **Job F: Run sp_BlitzCache (top problematic queries)**

Runs at **03:15 AM**

```
EXEC sp_BlitzCache @Top = 50, @ExpertMode = 1, @OutputTable = 'BlitzCacheOutput';
```

✓ **Job G: Query Store Performance Extract**

Runs at **03:30 AM**

```
INSERT INTO FactQueryStorePerf
SELECT
   SYSDATETIME(),
   S.ServerId,
   D.DatabaseId,
   Q.QueryId,
   rs.avg_duration,
   rs.avg_cpu_time,
   rs.avg_logical_io_reads,
   rs.count_executions
FROM sys.query_store_runtime_stats rs
JOIN sys.query_store_plan p ON rs.plan_id = p.plan_id
JOIN sys.query_store_query q ON p.query_id = q.query_id;
```

(We map QueryHash to DimQuery earlier.)

✓ **Job H: ETL from Repository → Data Warehouse**

Runs at **03:45 AM**

Example ETL:

Load Wait Stats

```
INSERT INTO FactWaitStats
SELECT
   R.RunId,
   S.ServerId,
   W.WaitType,
   W.WaitTimeMs,
   W.SignalWaitPct,
   R.RunTimestamp
FROM PSBlitzRepository..WaitStats W
JOIN PSBlitzRepository..RunInfo R ON R.RunId = W.RunId
JOIN DimServer S ON S.ServerName = R.ServerName;
```

Load Deadlocks

```
INSERT INTO FactDeadlocks(ServerId, Timestamp, Victim, LockMode, Resource, QueryId, XMLData)
SELECT
   S.ServerId,
   R.RunTimestamp,
   D.VictimProcess,
   D.LockMode,
   D.Resource,
   Q.QueryId,
   D.FullXML
FROM PSBlitzRepository..DeadlockEvents D
JOIN PSBlitzRepository..RunInfo R ON D.RunId = R.RunId
LEFT JOIN DimQuery Q ON Q.QueryText LIKE '%' + D.QueryText + '%'
JOIN DimServer S ON S.ServerName = R.ServerName;
```


🟩 **3. Retention Policies**

Add a job:

Keep 90 days of detail:

```
DELETE FROM PSBlitzRepository..WaitStats WHERE RunId < (SELECT MAX(RunId) - 90);
DELETE FROM ExecPlans WHERE PlanId < (SELECT MAX(PlanId) - 90);
DELETE FROM DeadlockEvents WHERE DeadlockId < (SELECT MAX(DeadlockId) - 90);
```

Keep 1 year of DW data:

```
DELETE FROM FactQueryStorePerf WHERE CaptureTime < DATEADD(YEAR, -1, SYSDATETIME());
```

🟦 **4. Power BI Dashboard (Model Summary)**

Pages:

1️⃣ **Server Health Overview**

- Warnings from sp_Blitz
- Deadlocks count

- Max wait stats for last 24h
- Perf trends

2 **Query Performance**

- Top queries by CPU / duration
- Plan regressions
- Query Store timeline

3 **Index Health**

- Fragmentation
- Missing indexes
- Duplicate/unused indexes

4 **Blocking & Deadlocks**

- Victims
- Lock resources
- Deadlock XML visualization

5 **Wait Stats Trend Explorer**

- Top N waits by day
- Signal vs. resource waits timelines

6 **Plan Issues**

- Plans with warnings
- Missing indexes from plans
- Parallelism issues

🟩 **5. Optional: Email Reports (6:00 AM)**

Send top-level health summary to DBA team:

Example:

```
EXEC msdb.dbo.sp_send_dbmail
  @profile_name = 'DBA Email',
  @recipients = 'dba-team@company.com',
  @subject = 'Nightly SQL Health Summary',
  @body = 'PSBlitz, BlitzCache, Query Store, and Deadlock reports updated.',
  @query = 'SELECT TOP 20 * FROM FactWaitStats ORDER BY WaitMs DESC';
```