Formation SQLServer PHENIX corrige (1)

Formation SQL Server

Administration et maintenance de la base PHENIX

Guide pratique — versions, configuration, diagnostic, performance, index et maintenance

TSI

Sommaire

Chapitre 1 — Présentation de l'environnement SQL Server

> 1.1 Identifier la version SQL Server
>
> 1.2 Informations serveur
>
> 1.3 Vérifier les bases disponibles
>
> 1.4 Faire évoluer le niveau de compatibilité
>
> 1.5 Mettre à jour les statistiques après le changement
>
> 1.6 Démarche recommandée avant un changement de niveau de compatibilité

Chapitre 2 — Vérification de la configuration SQL Server

> 2.1 Mémoire SQL Server
>
> 2.2 Processeur SQL Server
>
> 2.3 MAXDOP (Max Degree of Parallelism)
>
> 2.4 TempDB

Chapitre 3 — Diagnostic des incidents avec les logs

3.1 SQL Server Error Log

3.2 Les erreurs que tout administrateur doit connaître

3.3 SQL Server Agent Log

Chapitre 4 — Performance SQL Server

4.1 Requêtes les plus coûteuses (historique)

4.2 Détecter les blocages

Chapitre 5 — Index et statistiques

5.1 Mesurer la fragmentation (base PHENIX)

5.2 Décider de l'action à appliquer

5.3 Mettre à jour les statistiques

Chapitre 6 — Plan de maintenance de la base PHENIX

6.1 Planning hebdomadaire

6.2 Les tâches automatiques (Maintenance Plan)

6.3 Synthèse — planning recommandé pour PHENIX

Chapitre 7 — SHRINK DATABASE : bonnes et mauvaises pratiques

7.1 Pourquoi éviter le SHRINK systématique

7.2 Cas d'usage exceptionnels

Chapitre 1 — Présentation de l'environnement SQL Server

Objectif : *comprendre ce qui existe réellement sur le serveur avant toute intervention.*

1.1 Identifier la version SQL Server

```
SELECT
SERVERPROPERTY('ProductVersion') AS Version,
SERVERPROPERTY('ProductLevel') AS Niveau,
SERVERPROPERTY('Edition') AS Edition,
SERVERPROPERTY('ProductMajorVersion') AS VersionMajeure,
CASE CAST(SERVERPROPERTY('ProductMajorVersion') AS INT)
WHEN 16 THEN 'SQL Server 2022'
WHEN 15 THEN 'SQL Server 2019'
WHEN 14 THEN 'SQL Server 2017'
WHEN 13 THEN 'SQL Server 2016'
WHEN 12 THEN 'SQL Server 2014'
WHEN 11 THEN 'SQL Server 2012'
WHEN 10 THEN 'SQL Server 2008 / 2008 R2'
ELSE 'Version inconnue'
END AS VersionSQLServer;
```

Exemple de résultat :

Paramètre	Valeur
Version	16.0.xxxx
Edition	Standard
Version majeure	16 (SQL Server 2022)

Depuis SSMS (interface graphique)
Object Explorer → clic droit sur le nom du serveur → Properties → onglet General : affiche la version, l'édition et le nom du produit sans écrire de requête.

⚠ **Point de vigilance**
Vérifier systématiquement l'édition (Standard vs Enterprise) : plusieurs fonctionnalités utilisées plus loin dans ce guide — REBUILD d'index ONLINE, partitionnement, compression de données — ne sont disponibles qu'en édition Enterprise. Une commande copiée depuis un serveur Enterprise peut échouer sur un serveur Standard.

1.2 Informations serveur

```
SELECT
SERVERPROPERTY('MachineName') AS Serveur,
SERVERPROPERTY('ServerName') AS Instance,
SERVERPROPERTY('Collation') AS Collation;
```

Bonne pratique
Noter la collation par défaut de l'instance. Une collation différente entre l'instance, une base applicative et tempdb peut provoquer des erreurs de comparaison ("Cannot resolve collation conflict") dans les requêtes qui joignent des objets de plusieurs bases.

1.3 Vérifier les bases disponibles

```
SELECT
name,
state_desc,
recovery_model_desc,
compatibility_level
FROM sys.databases;
```

Colonne	Signification
state_desc = ONLINE	Base disponible et accessible
recovery_model_desc	Stratégie de sauvegarde (FULL, SIMPLE, BULK_LOGGED)
compatibility_level	Version de comportement du moteur SQL utilisée par la base

Depuis SSMS (interface graphique)
Object Explorer → dossier Databases : la couleur/icône de chaque base indique son état (une icône grisée avec une flèche signale souvent un état autre que ONLINE, ex. Restoring, Suspect).
Clic droit sur une base → Properties → Options : affiche Recovery Model et Compatibility Level.

⚠ **Point de vigilance**
Une base dans un état autre que ONLINE (SUSPECT, RECOVERY_PENDING, ou RESTORING de façon prolongée) doit être investiguée immédiatement : elle est inaccessible pour l'application tant que l'état n'est pas résolu.

1.4 Faire évoluer le niveau de compatibilité

Exemple pour passer du niveau 130 au niveau 160 :

```
ALTER DATABASE PHENIX SET COMPATIBILITY_LEVEL = 160;
```

Bonne pratique
Après le changement, il est recommandé de vider le cache des plans d'exécution afin que les requêtes soient recompilées avec le nouvel optimiseur, par exemple :

```
ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE_CACHE;
```

⚠ **Point de vigilance**
Cette commande provoque la recompilation des requêtes lors de leur prochaine exécution. Elle doit donc être exécutée pendant une fenêtre de maintenance.

1.5 Mettre à jour les statistiques après le changement

Faut-il aussi mettre à jour les statistiques ? Oui, c'est recommandé, surtout si les statistiques sont anciennes.

```
USE PHENIX;
GO
EXEC sp_updatestats;
GO
```

Des statistiques à jour permettent au nouvel optimiseur de faire de meilleurs choix.

1.6 Démarche recommandée avant un changement de niveau de compatibilité

Pour une base ERP de 70 Go :

1. Sauvegarde complète.
2. Vérification de l'intégrité (DBCC CHECKDB).
3. Mise à jour des statistiques (sp_updatestats).
4. Passage en COMPATIBILITY_LEVEL = 160.
5. Vider le cache des plans (pendant la maintenance).
6. Tester les fonctionnalités critiques de l'application.
7. Surveiller pendant quelques jours : CPU, requêtes lentes, blocages, temps de réponse.

Chapitre 2 — Vérification de la configuration SQL Server

Objectif : *savoir si SQL Server est correctement configuré pour la charge de travail réelle.*

2.1 Mémoire SQL Server

Vérifier la mémoire utilisée

```
SELECT name, value_in_use
FROM sys.configurations
WHERE name LIKE '%max server memory%';
```

Exemple : max server memory = 65536 Mo → SQL Server peut utiliser au maximum 64 Go.

Modifier la mémoire maximale (T-SQL)

```
EXEC sp_configure 'show advanced options', 1;
RECONFIGURE;
EXEC sp_configure 'max server memory', 65536;
RECONFIGURE;
```

Depuis SSMS (interface graphique)
Object Explorer → clic droit sur l'instance → Properties → page Memory.
Renseigner « Maximum server memory (in MB) », cliquer sur OK — aucun redémarrage du service n'est nécessaire.

Bonne pratique
Laisser toujours de la mémoire disponible pour l'OS (généralement 4 à 8 Go, plus selon la RAM totale et les autres services installés sur la machine).

2.2 Processeur SQL Server

```
SELECT
cpu_count AS [Nombre de processeurs logiques détectés],
scheduler_count AS [Nombre de processeurs utilisés par SQL Server],
(cpu_count - scheduler_count) AS [Nombre de processeurs non utilisés par SQL Server],
socket_count AS [Nombre de sockets (processeurs physiques)],
cores_per_socket AS [Nombre de cœurs par socket],
hyperthread_ratio AS [Ratio Hyper-Threading]
FROM sys.dm_os_sys_info;
```

Exemple d'interprétation : cpu_count = 8, socket_count = 1, cores_per_socket = 4 → calcul : Socket × Core × HyperThread = 1 × 4 × 2 = 8 CPU logiques.

⚠ **Point de vigilance**
Si cpu_count est très inférieur au nombre de cœurs physiques réels de la machine, cela peut indiquer un affinity mask résiduel ou une licence limitée — à vérifier systématiquement après une migration ou un upgrade in-place.

Depuis SSMS (interface graphique)
Object Explorer → clic droit sur l'instance → Properties → page Processors : affiche les cœurs disponibles, le masque d'affinité (Processor Affinity) et l'affinité I/O.
C'est ici que se corrige un affinity mask résiduel bloquant l'utilisation de tous les cœurs après une migration.

2.3 MAXDOP (Max Degree of Parallelism)

```
SELECT name, value_in_use
FROM sys.configurations
WHERE name = 'max degree of parallelism';
```

Bonne pratique
Règle générale (Microsoft) : ne pas dépasser 8, et l'aligner sur le nombre de cœurs par nœud NUMA ; à ajuster selon la charge réelle (OLTP vs data warehouse).

Depuis SSMS (interface graphique)
Object Explorer → clic droit sur l'instance → Properties → page Advanced → section « Parallelism » : champs Max Degree of Parallelism et Cost Threshold for Parallelism.

Seuil de coût pour le parallélisme (Cost Threshold for Parallelism)

```
SELECT
name AS [Paramètre],
value AS [Valeur configurée],
value_in_use AS [Valeur utilisée]
FROM sys.configurations
WHERE name = 'cost threshold for parallelism';
```

⚠ **Point de vigilance**
La valeur par défaut de SQL Server est 5 — une valeur héritée des années 1998, sur du matériel largement dépassé aujourd'hui. Sur un serveur moderne, une valeur de 5 est ridiculement basse : énormément de requêtes banales dépassent ce seuil et se retrouvent parallélisées inutilement, ce qui gaspille du CPU et peut même ralentir le serveur global.

Valeur	Contexte
5 (défaut)	À ne jamais garder tel quel en production
25 à 50	Recommandation courante pour un serveur moderne, point de départ raisonnable
Plus haut (ex. 75-100)	Si vous constatez encore trop de parallélisme sur des requêtes qui devraient rester simples

```
EXEC sp_configure 'cost threshold for parallelism', 50;
RECONFIGURE;
```

2.4 TempDB

```
USE tempdb;
SELECT
name,
size*8/1024 AS SizeMB,
growth*8/1024 AS GrowthMB
FROM sys.database_files;
```

Points à contrôler :

- Nombre de fichiers de données (idéalement plusieurs fichiers de taille identique sur des serveurs multi-cœurs)
- Taille identique entre tous les fichiers de données
- Croissance fixe en Mo, jamais en pourcentage

Depuis SSMS (interface graphique)
Object Explorer → Databases → System Databases → tempdb → clic droit → Properties → page Files : liste des fichiers, taille, et croissance (Autogrowth / Maxsize) modifiables directement dans la grille.

Nombre de fichiers TempDB recommandé

La règle Microsoft n'est pas « 1 fichier par cœur » — c'est une erreur courante. La recommandation officielle est la suivante :

Cœurs logiques	Fichiers de données TempDB recommandés
≤ 8 cœurs	= nombre de cœurs
> 8 cœurs (ex. 20 cœurs)	8 fichiers pour commencer, à augmenter par paliers de 4 seulement si contention constatée

Une configuration avec tempdev + temp2 à temp8 (8 fichiers de données) sur un serveur à 20 cœurs est donc déjà correcte. Pas besoin de passer à 20 fichiers — au-delà de 8, le gain devient marginal et complique la gestion, sauf preuve réelle de contention.

Comment vérifier s'il faut aller au-delà de 8 fichiers

```
SELECT wait_type, waiting_tasks_count, wait_time_ms
FROM sys.dm_os_wait_stats
WHERE wait_type LIKE 'PAGELATCH%';
```

Si PAGELATCH_UP (ou PAGELATCH_EX) accumule un temps d'attente élevé et en croissance, c'est le signal pour ajouter 4 fichiers de plus. Sans ce signal, restez à 8.

Recommandation concrète

Fichier	Taille initiale recommandée	Croissance recommandée
tempdev, temp2…temp8 (8 fichiers données)	512 Mo à 1 Go chacun (à ajuster selon l'usage réel observé)	256 Mo à 512 Mo fixe (jamais en %)
templog	512 Mo à 1 Go	256 Mo fixe

```
ALTER DATABASE tempdb MODIFY FILE (NAME = tempdev, SIZE = 1024MB, FILEGROWTH = 256MB);
ALTER DATABASE tempdb MODIFY FILE (NAME = temp2, SIZE = 1024MB, FILEGROWTH = 256MB);
-- répéter à l'identique pour temp3 à temp8
ALTER DATABASE tempdb MODIFY FILE (NAME = templog, SIZE = 1024MB, FILEGROWTH = 256MB);
```

 Bonne pratique
Règle d'or à garder : tous les fichiers de données doivent rester strictement identiques en taille et en croissance — sinon l'algorithme de répartition de charge entre fichiers (proportional fill) devient inefficace et la contention réapparaît.

Chapitre 3 — Diagnostic des incidents avec les logs

Objectif : *ne pas apprendre à lire toutes les lignes des journaux, mais à retrouver rapidement les événements importants.*

3.1 SQL Server Error Log

Dans SSMS : Management → SQL Server Logs → Current (et Archive #1, #2… pour l'historique).

Méthode 1 — Filtrage dans SSMS

Ouvrir le journal Current, puis clic droit → Filter, et rechercher un mot-clé : Error, failed, severity, backup, I/O, login failed.

Méthode 2 — sp_readerrorlog (recommandée pour l'automatisation)

```
EXEC sp_readerrorlog; -- journal courant complet
```

Recherches ciblées utiles au quotidien :

```
-- Échecs de connexion
EXEC sp_readerrorlog 0, 1, 'Login failed';
-- Sauvegardes
EXEC sp_readerrorlog 0, 1, 'Backup';
-- Erreurs disque / I-O
EXEC sp_readerrorlog 0, 1, 'I/O';
-- Corruption
EXEC sp_readerrorlog 0, 1, '823';
EXEC sp_readerrorlog 0, 1, '824';
EXEC sp_readerrorlog 0, 1, '825';
-- Transaction log plein
EXEC sp_readerrorlog 0, 1, '9002';
-- Espace fichier insuffisant
EXEC sp_readerrorlog 0, 1, '1105';
```

 Bonne pratique
Automatiser une recherche quotidienne des mots-clés critiques (823, 824, 825, 9002, Login failed) via un job SQL Server Agent avec alerte, plutôt qu'une consultation manuelle — pour ne pas dépendre de la vigilance de l'administrateur ce jour-là.

3.2 Les erreurs que tout administrateur doit connaître

Erreur	Signification	Gravité
823	Erreur d'entrée/sortie (I/O)	Critique
824	Corruption logique ou physique	Critique
825	Relecture de page nécessaire	Très critique — souvent un signe de stockage défaillant
9002	Journal des transactions plein	Important
1105	Impossible d'allouer de l'espace	Important
Login failed	Tentative de connexion échouée	À surveiller
Deadlock	Blocage entre transactions	À analyser

⚠ **Point de vigilance**
Les erreurs de sévérité 20 et plus provoquent la coupure automatique de la connexion côté client et sont toujours écrites dans l'Error Log : elles doivent déclencher une investigation immédiate, pas seulement une note pour plus tard.

3.3 SQL Server Agent Log

Le Error Log du moteur SQL Server enregistre les événements du moteur ; le SQL Server Agent Log enregistre les événements liés aux jobs (sauvegardes, maintenance, etc.). Chemin : SQL Server Agent → Error Logs.

Chapitre 4 — Performance SQL Server

Objectif : *identifier pourquoi une application est lente, avec une méthode reproductible.*

4.1 Requêtes les plus coûteuses (historique)

```
SELECT TOP 20
DB_NAME(st.dbid) AS [Base de données],
qs.execution_count AS [Nombre exécutions],
CAST(
qs.total_elapsed_time * 1.0
/ NULLIF(qs.execution_count,0)
/ 1000
AS DECIMAL(18,2)) AS [Temps moyen ms],
qs.last_execution_time AS [Dernière exécution],
qs.total_logical_reads AS [Lectures mémoire],
qs.total_logical_writes AS [Ecritures],
qs.total_worker_time / 1000 AS [CPU total ms],
LEFT(st.text, 500) AS [Requête SQL]
FROM sys.dm_exec_query_stats qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) st
ORDER BY [Temps moyen ms] DESC;
```

⚠ **Point de vigilance**
Les compteurs de sys.dm_exec_query_stats sont cumulés depuis le dernier redémarrage du service ou la dernière purge du cache de plans : un total élevé peut refléter l'historique plutôt que la charge actuelle. Toujours croiser avec last_execution_time avant de conclure.

4.2 Détecter les blocages

```
SELECT
blocking_session_id,
wait_type,
wait_time,
status
FROM sys.dm_exec_requests
WHERE blocking_session_id <> 0;
```

Un blocage n'est pas anormal en soi (les transactions concurrentes se bloquent naturellement) ; il devient un problème quand il persiste dans la durée ou provoque des deadlocks répétés.

◫ **Depuis SSMS (interface graphique)**
Clic droit sur le serveur → Activity Monitor → volet Processes : trier par la colonne « Blocked By » pour repérer immédiatement les sessions bloquantes et bloquées, sans écrire de requête.

Chapitre 5 — Index et statistiques

Objectif : *maintenir les performances dans le temps en gérant la fragmentation et les statistiques.*

5.1 Mesurer la fragmentation (base PHENIX)

```
USE PHENIX;
SELECT
OBJECT_NAME(object_id) AS TableName,
index_id,
avg_fragmentation_in_percent,
page_count
FROM sys.dm_db_index_physical_stats
(DB_ID(), NULL, NULL, NULL, 'LIMITED')
WHERE page_count > 1000
ORDER BY avg_fragmentation_in_percent DESC;
```

Le filtre page_count > 1000 exclut les petits index dont la fragmentation n'a quasiment aucun impact en pratique.

Depuis SSMS (interface graphique)
Clic droit sur la base PHENIX → Reports → Standard Reports → « Index Physical Statistics » : affiche la fragmentation de tous les index sous forme de tableau, équivalent visuel de la requête ci-dessus.

5.2 Décider de l'action à appliquer

Fragmentation	Action
< 5 %	Aucune action
5 % – 30 %	REORGANIZE
> 30 %	REBUILD

REORGANIZE

```
ALTER INDEX ALL ON dbo.TableName REORGANIZE;
```

REBUILD

```
ALTER INDEX ALL ON dbo.TableName REBUILD;
```

Depuis SSMS (interface graphique)
Object Explorer → base → Tables → table concernée → Indexes → clic droit sur un index (ou sur le dossier Indexes pour tous les traiter) → Reorganize… ou Rebuild… : ouvre une boîte de dialogue équivalente aux commandes ALTER INDEX. Pour traiter toute une base en une fois : clic droit sur la base → Tasks → Rebuild Index… (assistant Reorganize/Rebuild avec sélection multiple de tables).

Script pour traiter tout automatiquement (au lieu de taper 100+ ALTER INDEX à la main)

1) Script REORGANIZE (fragmentation entre 5 % et 30 %)

```
USE PHENIX;
GO
DECLARE @TableName NVARCHAR(255);
DECLARE @IndexName NVARCHAR(255);
DECLARE @SQL NVARCHAR(MAX);
DECLARE IndexCursor CURSOR FOR
SELECT
OBJECT_SCHEMA_NAME(ips.object_id) + '.' + OBJECT_NAME(ips.object_id) AS TableName,
i.name AS IndexName
FROM sys.dm_db_index_physical_stats(DB_ID('PHENIX'), NULL, NULL, NULL, 'LIMITED') ips
INNER JOIN sys.indexes i
ON ips.object_id = i.object_id
AND ips.index_id = i.index_id
WHERE
ips.page_count > 1000
AND ips.avg_fragmentation_in_percent BETWEEN 5 AND 30
AND i.type > 0
AND i.name IS NOT NULL;
OPEN IndexCursor;
FETCH NEXT FROM IndexCursor INTO @TableName, @IndexName;
WHILE @@FETCH_STATUS = 0
BEGIN
SET @SQL = 'ALTER INDEX [' + @IndexName + '] ON ' + @TableName + ' REORGANIZE;';
PRINT @SQL;
EXEC sp_executesql @SQL;
FETCH NEXT FROM IndexCursor INTO @TableName, @IndexName;
END;
CLOSE IndexCursor;
DEALLOCATE IndexCursor;
GO
```

2) Script REBUILD (fragmentation > 30 %)

```
USE PHENIX;
GO
DECLARE @TableName NVARCHAR(255);
DECLARE @IndexName NVARCHAR(255);
DECLARE @SQL NVARCHAR(MAX);
DECLARE IndexCursor CURSOR FOR
SELECT
OBJECT_SCHEMA_NAME(ips.object_id) + '.' + OBJECT_NAME(ips.object_id),
i.name
FROM sys.dm_db_index_physical_stats(DB_ID('PHENIX'), NULL, NULL, NULL, 'LIMITED') ips
INNER JOIN sys.indexes i
ON ips.object_id = i.object_id
AND ips.index_id = i.index_id
WHERE
ips.page_count > 1000
AND ips.avg_fragmentation_in_percent > 30
AND i.type > 0
AND i.name IS NOT NULL;
OPEN IndexCursor;
FETCH NEXT FROM IndexCursor INTO @TableName, @IndexName;
WHILE @@FETCH_STATUS = 0
BEGIN
SET @SQL = 'ALTER INDEX [' + @IndexName + '] ON ' + @TableName + ' REBUILD;';
PRINT @SQL;
EXEC sp_executesql @SQL;
FETCH NEXT FROM IndexCursor INTO @TableName, @IndexName;
END;
CLOSE IndexCursor;
DEALLOCATE IndexCursor;
GO
```

3) Version recommandée pour SQL Server 2022 (production)

Sur SQL Server 2022 Enterprise, vous pouvez améliorer le REBUILD :

```
ALTER INDEX [Nom_Index]
ON [Schema].[Table]
REBUILD WITH
(
ONLINE = ON,
SORT_IN_TEMPDB = ON,
MAXDOP = 4
);
```

Option	Rôle
ONLINE = ON	Les utilisateurs continuent à travailler pendant le rebuild
SORT_IN_TEMPDB = ON	Utilise TempDB pour le tri, souvent plus rapide
MAXDOP = 4	Limite l'utilisation CPU

⚠ **Point de vigilance**
ONLINE = ON dépend de l'édition SQL Server et du type d'index.

5.3 Mettre à jour les statistiques

```
USE PHENIX;
EXEC sp_updatestats;
```

Après un REORGANIZE, la mise à jour des statistiques est recommandée. Après un REBUILD, elle n'est généralement pas nécessaire pour les index reconstruits (les statistiques sont recalculées avec un taux d'échantillonnage à 100 %).

Depuis SSMS (interface graphique)
Clic droit sur la base → Tasks → Update Statistics… : choisir « All existing statistics » et le type de scan (Full scan / Sample).

Chapitre 6 — Plan de maintenance de la base PHENIX

Objectif : *disposer d'un planning de maintenance clair, applicable immédiatement sur PHENIX.*

6.1 Planning hebdomadaire

1. Contrôle d'intégrité

```
DBCC CHECKDB('PHENIX');
```

Détecte les corruptions, erreurs de pages et problèmes de structure. À exécuter en dehors des heures de forte charge.

En cas de corruption :

La bonne pratique — l'ordre à respecter

8. Ne jamais lancer une réparation en réflexe dès qu'une corruption est détectée.
9. La vraie solution en cas de corruption est de restaurer depuis une sauvegarde saine (backup Full + logs), pas de réparer sur place.
10. REPAIR_ALLOW_DATA_LOSS = solution de dernier recours, seulement si aucune sauvegarde exploitable n'existe.
11. Il faut mettre la base en SINGLE_USER, ce qui coupe tous les utilisateurs connectés — donc à faire uniquement en fenêtre de maintenance, jamais en production active.

Les deux options de réparation, et pourquoi elles sont dangereuses :

Option	Ce qu'elle fait	Risque
REPAIR_REBUILD	Corrige les erreurs sans perte de données (reconstruction d'index principalement)	Faible risque, mais ne résout pas tout
REPAIR_ALLOW_DATA_LOSS	Corrige en supprimant les pages/lignes corrompues	Perte de données réelle et définitive sur les données corrompues

Depuis SSMS (interface graphique)
Clic droit sur la base → Tasks → Check Database Integrity… : coche « Include indexes » pour un contrôle complet, résultat affiché dans une fenêtre de rapport.

2. Analyse et maintenance des index

Selon la fragmentation mesurée (voir chapitre 5).

3. Mise à jour des statistiques

```
EXEC sp_updatestats;
```

4. Croissance des fichiers

```
SELECT
name,
size*8/1024 AS SizeMB,
growth*8/1024 AS GrowthMB,
is_percent_growth
FROM sys.database_files;
```

Vérifier : pas de croissance en pourcentage, une croissance fixe, une taille cohérente avec l'usage réel.

Recommandation : Growth : 512 MB ou 1024 MB.

Bonne pratique
Documenter et tester la procédure de restauration régulièrement, pas seulement vérifier que les sauvegardes se terminent sans erreur : une sauvegarde jamais restaurée en test n'est pas une garantie de récupération.

6.2 Les tâches automatiques (Maintenance Plan)

Un plan de maintenance complet suit un ordre précis :

12. Integrity Check (Check Database Integrity)
13. Index Maintenance (Reorganize / Rebuild selon fragmentation)
14. Update Statistics
15. Cleanup History
16. Backup

Depuis SSMS (interface graphique)
Object Explorer → Management → Maintenance Plans → clic droit → New Maintenance Plan… (ou « Maintenance Plan Wizard » pour une création guidée).
Glisser-déposer les tâches dans l'ordre voulu (Check DB Integrity → Rebuild/Reorganize Index → Update Statistics → History Cleanup → Back Up Database), les relier par des flèches, puis définir la planification (Subplan Schedule).

6.3 Synthèse — planning recommandé pour PHENIX

Fréquence	Tâche	Méthode
Quotidien	Vérifier les jobs	SQL Server Agent
Quotidien	Vérifier les sauvegardes	Requête SQL + Jobs
Quotidien	Vérifier l'Error Log	SSMS / sp_readerrorlog
Hebdomadaire	CHECKDB	DBCC CHECKDB
Hebdomadaire	Mesurer la fragmentation	sys.dm_db_index_physical_stats
Hebdomadaire	REORGANIZE (5-30 %)	ALTER INDEX ... REORGANIZE
Hebdomadaire	Update statistics	sp_updatestats
Mensuel	REBUILD (> 30 %)	ALTER INDEX ... REBUILD
Exceptionnel	SHRINK	Uniquement avec un besoin justifié

Chapitre 7 — SHRINK DATABASE : bonnes et mauvaises pratiques

Le SHRINK ne doit jamais faire partie d'un plan de maintenance automatisé.

7.1 Pourquoi éviter le SHRINK systématique

```
DBCC SHRINKDATABASE(PHENIX); -- à ne pas planifier
```

- Il augmente fortement la fragmentation des index
- Il provoque une charge I/O importante pendant son exécution
- Le fichier a tendance à regrossir rapidement, créant un cycle contre-productif

Depuis SSMS (interface graphique)
Clic droit sur la base → Tasks → Shrink → Database (ou Files pour cibler un fichier précis) : à n'utiliser que dans les cas exceptionnels décrits ci-dessous, jamais en tâche planifiée.

7.2 Cas d'usage exceptionnels

- Suppression de plusieurs centaines de Go de données (purge, archivage massif)
- Erreur ayant fait grossir anormalement le fichier de log
- Besoin réel et ponctuel de récupérer de l'espace disque

```
DBCC SHRINKFILE ('PHENIX_Data', 50000);
```

⚠ **Point de vigilance**
Important : après un SHRINK des fichiers de données, il est généralement nécessaire de reconstruire les index concernés, car l'opération augmente leur fragmentation.

Revision #1
Created 2026-10-28 13:11:58 UTC by bookstack
Updated 2026-10-28 13:11:58 UTC by bookstack