Projet de recherche
Documentation SQL Server

Master 1 MIAGE, Université Côte d'Azur

Année Universitaire 2023-2024

Cours de Fonctionnement d'un SGBD, G.Galli

Réalisé par :

Youssef ANSARI
Guillaume BENEZECH LOUSTALOT FOREST
Sawsen EL BAHRI
Axel VERKIMPE
Noémie POUJOL

Sommaire

1. Identification du SGBD

Nom SGBD	**SQL Server**
Date de parution	Première version : 24 avril **1989** Dernière version : 16 novembre **2022**
Créateur du SGBD	Initialement codéveloppé par **Sybase** et **Microsoft**, **Ashton-Tate** a également été associé à sa première version
Propriétaire	**Microsoft**
Modèles de données supportés	Modèle **relationnel** Modèle **hiérarchique** Modèle **XML** Modèle **spatial** Modèle **en colonnes**
Licence	Licence propriétaire et EULA
Écrit en	C++, C et C#
Système d'exploitation	Linux, Microsoft Windows et **Windows Server**
Type	**Système de gestion de base de données relationnelle** (SGBDR) Protocole de communication Organisation
Site web	www.microsoft.com/sql-server

2. Architecture fonctionnelle

2.1 Architecture

SQL Server est basé sur un modèle client-serveur et son moteur de base de données se concentre sur la gestion du stockage, l'accès aux données, et la gestion des transactions. Il intègre également un Query Optimizer qui optimise les requêtes pour améliorer les performances. Ainsi, les clients établissent des connexions au serveur SQL afin d'envoyer des requêtes et de recevoir les résultats correspondants.

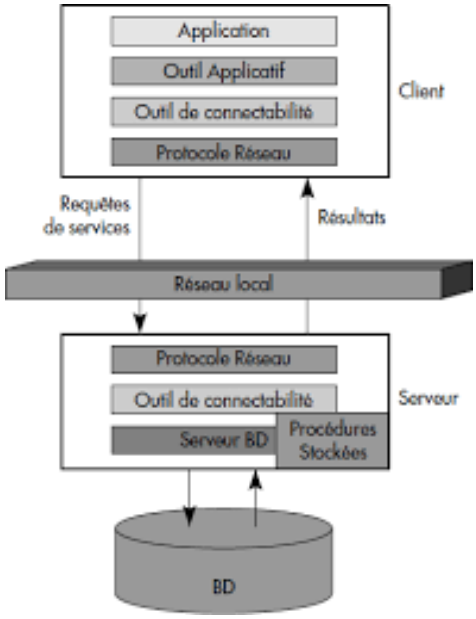

Figure 1- Architecture de communication entre la base de données et le client

2.2 Différents caches mémoires et leurs rôles

Il existe plusieurs caches dans le SGBD qui optimisent ses performances en réduisant les accès aux disques durs. Ces caches améliorent l'efficacité des opérations de récupération de données et minimisent les coûts liés à l'optimisation des requêtes. Voici quelques-uns de ces caches :

- **Cache de requêtes :** Il stocke les plans de requêtes afin de réduire la nécessité de réoptimiser les requêtes identiques.
- **Cache de plan de requête :** Il contient les plans d'exécution des requêtes optimisées, c'est-à-dire, comment les requêtes seront exécutées de manière à réduire le temps et les ressources déployées.
- **Cache de procédures stockées :** Il stocke les plans d'exécution des procédures stockées pour améliorer leur réutilisation.
- **Cache de mémoire intermédiaire (Buffer Cache) :** Il stocke des pages de données fréquemment utilisées en mémoire pour minimiser les accès des disques durs.
- **Cache des logs :** Il contient l'historique des logs, donc de toutes les requêtes qui se sont exécutées.
- **Cache de Pages de Données :** C'est le stockage en mémoire de pages de données spécifiques pour faciliter l'accès rapide à ces données. Les pages de données représentent des blocs de données physiques sur les disques durs.

2.3 Processus gravitant autour du cache mémoire et leur rôle

Dans le contexte de SQL Server, différents processus collaborent avec le cache mémoire pour améliorer les performances et minimiser les accès disque.

- **Query Optimizer :**

Rôle : C’est un composant clé du moteur SQL Server, son rôle est d'analyser les requêtes SQL soumises au serveur et de déterminer le plan d'exécution le plus efficace pour récupérer les données demandées

Relation avec la cache mémoire : Le Query Optimizer prend en compte le cache de requêtes et le cache de plan de requête, S'il trouve un plan de requête identique déjà présent dans le cache, il peut réutiliser ce plan plutôt que de le recalculer.

- **Buffer Manager :**

Rôle : Gère le Buffer Cache et s'assure que les pages de données fréquemment utilisées sont stockées en RAM pour minimiser les accès au stockage sur disque.

Relation avec la cache mémoire : Les pages de données mises en cache par le Buffer Manager viennent de différentes tables et index, et ce cache contribue à réduire les temps d'accès aux données en mémoire.

- **Gestionnaire de Verrous (Lock Manager) :**

Rôle : Contrôle l'accès concurrent aux données. Il assure l'isolation des transactions et prévient les conflits d'accès concurrents.

Relation avec la cache mémoire : Le Lock Manager garantit la cohérence des données, et il peut également interagir avec le cache mémoire pour garantir la validité des données accédées.

- **Database Engine :**

Rôle : Gère l'ensemble du processus d'exécution des requêtes, y compris l'interprétation des plans de requêtes générés par le Query Optimizer, l'accès aux données dans le Buffer Cache, l'application de verrous, …

Relation avec la cache mémoire : Le Database Engine utilise le cache mémoire pour minimiser les accès disque et optimiser l'accès aux données.

2.4 Gestion des connexions

Lorsqu'on aborde les opérations sur une base de données, deux modes d'exécution prédominants sont utilisés : le mode ligne et le mode batch. Chacun de ces modes présente des caractéristiques distinctes en termes de gestion des connexions, du pool de connexions, de la gestion des sessions, de la limitation des connexions, et de la déconnexion.

- **Connexions en Mode Ligne**

Définition *:* En mode ligne, les opérations sont effectuées sur une ligne à la fois. Les transactions sont souvent de courte durée, impliquant des insertions, mises à jour ou suppressions d'enregistrements.

Gestion des Connexions *:* Chaque opération individuelle peut nécessiter une connexion distincte au serveur, notamment dans le cas de transactions OLTP (traitement transactionnel en ligne) où des opérations fréquentes sont effectuées.

- **Connexions en Mode Batch**

Définition : En mode batch, les opérations sont effectuées sur plusieurs lignes simultanément. Cela est particulièrement efficace pour des opérations analytiques sur de grandes quantités de données.

Gestion des Connexions : En mode batch, une connexion peut être utilisée pour traiter un ensemble de lignes en une seule opération. Les connexions peuvent être utilisées efficacement grâce à la nature simultanée des opérations en mode batch.

- **Liens avec ces deux modes**

Pool de connexions et optimisation : Le pool de connexions est utilisé dans les deux modes d'exécution. Il optimise les performances en utilisant les connexions existantes, que ce soit pour une série d'opérations en mode ligne ou pour une opération en mode batch.

Gestion des sessions : La gestion des sessions peut être influencée par le mode d'exécution. En mode ligne, chaque opération individuelle peut être considérée comme une session distincte, tandis qu'en mode batch, une session peut englober plusieurs opérations simultanées.

La limitation des connexions : La limitation des connexions peut être ajustée en fonction des besoins spécifiques du mode d'exécution. En mode batch, où des opérations massives peuvent être effectuées, une limitation plus élevée peut être justifiée.

Déconnexion : La gestion des déconnexions est importante dans les deux modes. Les connexions peuvent être automatiquement libérées après une opération en mode ligne ou après le traitement d'un lot en mode batch.

2.5 Processus d’exécution des requêtes

Le mode en Ligne et le mode Batch présentent des processus distincts du début à la fin de l'exécution des requêtes qui influencent la manière dont les données sont traitées. Voici une analyse comparative des processus d'exécution spécifiques à ces deux modes.

- **Processus d'Exécution des Requêtes en Mode Ligne :**

Requête Entrante : L'utilisateur envoie une requête SQL au serveur.

Analyse Syntaxique et Sémantique : Le moteur de base de données analyse la requête pour s'assurer qu'elle est correcte sur le plan syntaxique et sémantique.

Optimisation de Requête : Le Query Optimizer génère un plan d'exécution optimal pour la requête, en choisissant les indices appropriés, les opérations d'assemblage des résultats, etc.

Exécution du Plan : Le moteur de base de données exécute le plan d'exécution généré. En mode ligne, cela signifie que chaque ligne est lue et traitée une à une selon le plan.

Renvoi des Résultats : Les résultats de la requête sont renvoyés à l'utilisateur.

- **Processus d'Exécution des Requêtes en Mode Batch :**

Requête Entrante : L'utilisateur envoie une requête SQL au serveur.

Analyse Syntaxique et Sémantique : Le moteur de base de données analyse la requête pour s'assurer de sa validité.

Optimisation de Requête : Le Query Optimizer génère un plan d'exécution optimisé, mais en mode batch, ce plan peut impliquer le traitement de plusieurs lignes simultanément.

Exécution du Plan : Le moteur de base de données exécute le plan d'exécution généré. En mode batch, cela signifie que des groupes de lignes (batches) sont traités en même temps, exploitant le parallélisme et les optimisations spécifiques au traitement en mode batch.

Renvoi des Résultats : Les résultats de la requête sont renvoyés à l'utilisateur.

En comparaison, le mode en ligne est idéal dans le traitement de transactions rapides et fréquentes, assurant la cohérence des données avec l'utilisation d'indexes traditionnels. À l'inverse, le mode batch est optimal pour le traitement massif de données, exploitant le parallélisme et des optimisations spécifiques au traitement en mode batch, et potentiellement la compression de données. Il se distingue par son efficacité dans les opérations analytiques sur des ensembles de données étendus.

3. Le dictionnaire de données

Tables système :

SQL Server stocke les données qui définissent la configuration du serveur et de toutes ses tables dans un ensemble spécial de tables appelées « tables système ». Les utilisateurs ne peuvent pas directement interroger ou mettre à jour les tables système. Les informations contenues dans les tables système sont disponibles via les affichages système (vue système).

Les métadonnées dans les tables systèmes :

Un bénéficiaire disposant d'une autorisation CONTROL, ALTER ou VIEW DEFINITION sur une base de données peut voir les métadonnées de table de base système dans l'affichage catalogue sys.objects . Le bénéficiaire peut également résoudre les noms et les ID d'objet des tables de base système à l'aide de fonctions intégrées telles que OBJECT_NAME et OBJECT_ID.

Le tableau suivant répertorie et décrit quelques tables de base système dans SQL Server :

Table de base	Description
sys.sysschobjs	Existe dans toutes les bases de données. Chaque ligne représente un objet de la base de données.
sys.sysbinobjs	Existe dans toutes les bases de données. Contient une ligne pour chaque entité de Service Broker dans la base de données. Les entités de Service Broker incluent les éléments suivants : - Type de message - Contrat de service - Service Les noms et les types utilisent un classement binaire fixe.

sys.sysclsobjs	Existe dans toutes les bases de données. Contient une ligne pour chaque entité classifiée qui partage les mêmes propriétés communes qui incluent les éléments suivants : - Assembly - Unité de sauvegarde - Catalogue de texte intégral - Fonction de partition - Schéma de partition - Groupe de fichiers - Clé d'obfuscation - Schéma
sys.sysnsobjs	Existe dans toutes les bases de données. Contient une ligne pour chaque entité de l'étendue de l'espace de noms. Cette table est utilisée pour le stockage des entités de collection XML.
sys.syscolpars	Existe dans toutes les bases de données. Contient une ligne pour chaque colonne de table, chaque vue ou chaque fonction table. Contient également des lignes pour chaque paramètre d'une procédure ou d'une fonction.

Vues système :

Comme Microsoft a décidé de cacher les tables systèmes et de rendre leur accès difficile, à la place on utilise des vues de catalogue. Les vues système dans Microsoft SQL Server fournissent des interfaces virtuelles pour accéder aux métadonnées et aux informations sur la base de données. Ces vues sont souvent utilisées pour interroger le dictionnaire de données de SQL Server de manière plus conviviale que d'accéder directement aux tables système.

Les vues système exposent les métadonnées de catalogue. Vous pouvez utiliser des vues système pour retourner des informations sur l'instance de SQL Server ou les objets définis dans l'instance. Par exemple, vous pouvez interroger la vue de catalogue sys.databases pour retourner des informations sur les bases de données définies par l'utilisateur disponibles dans l'instance.

Il existe différents types de vues :

- **Vues de catalogue système (INFORMATION_SCHEMA) :**

Rôle : Ces vues fournissent des informations standard sur la structure de la base de données, telles que les tables, les colonnes, les clés étrangères, etc.
Exemple :

- *information_schema.tables* : Cette vue fournit des informations sur les tables de la base de données, telles que le nom de la table, le schéma, le type d'objet, etc.

- *information_schema.columns* : Contient des informations sur les colonnes de toutes les tables de la base de données, telles que le nom de la colonne, le type de données, etc.
- *information_schema.views* : Cette vue donne des détails sur les vues définies dans la base de données, y compris le nom de la vue, le schéma, etc.

- **Vues de compatibilité système (syscompatibilities) :**

Rôle : Historiquement utilisé pour des raisons de compatibilité entre différentes versions de SQL Server. Elles exposent les mêmes métadonnées que celles disponibles dans SQL Server 2000 (8.x). Toutefois, les vues de compatibilité n'exposent aucune des métadonnées liées aux fonctionnalités introduites dans SQL Server 2005 (9.x) et versions ultérieures.

Exemple :

- Sysc*ompatibilities*

- **Vues de gestion dynamique système (sys.dm_) :**

Rôle : Ces vues fournissent des informations en temps réel sur les performances et l'état du serveur, telles que les connexions, les verrous, l'utilisation de la mémoire, etc. Elles peuvent être utilisées pour surveiller l'intégrité d'une instance de serveur, diagnostiquer les problèmes et régler les performances.

Exemple :

- sys.*dm_exec_connections*
- *sys.dm_os_wait_stats*

- **Vues de schémas d'information système (sys.views) :**

Rôle : Les vues de schéma d'informations fournissent une vue interne indépendante de la table système des métadonnées SQL Server. Elles permettent aux applications de fonctionner correctement bien que des changements importants aient été apportés aux tables système sous-jacentes

Exemple :

- *sys.views*

4. Organisation physique

Chaque base de données SQL Server a au moins deux fichiers principaux : un fichier de données et un fichier journal. Les fichiers de données stockent les informations telles que les tables, les index, les procédures stockées et les vues. Les fichiers journaux enregistrent les données nécessaires pour récupérer toutes les transactions de la base de données.

Les bases de données SQL Server ont trois types de fichier :

- **Principal (.mdf)** : Il stocke les données principales de la base de données. Chaque base de données contient un fichier de données primaire.

- **Secondaire (.ndf)** : Ils sont utilisés pour stocker des données de manière additionnelle, la charge est répartie sur plusieurs fichiers. Les données sont réparties sur plusieurs disques en plaçant chaque fichier sur un lecteur disque distinct.

- **Journal des transactions (.ldf)** : Il enregistre les transactions effectuées dans la base de données pour permettre la récupération. Chaque base de données doit posséder au moins un fichier journal.

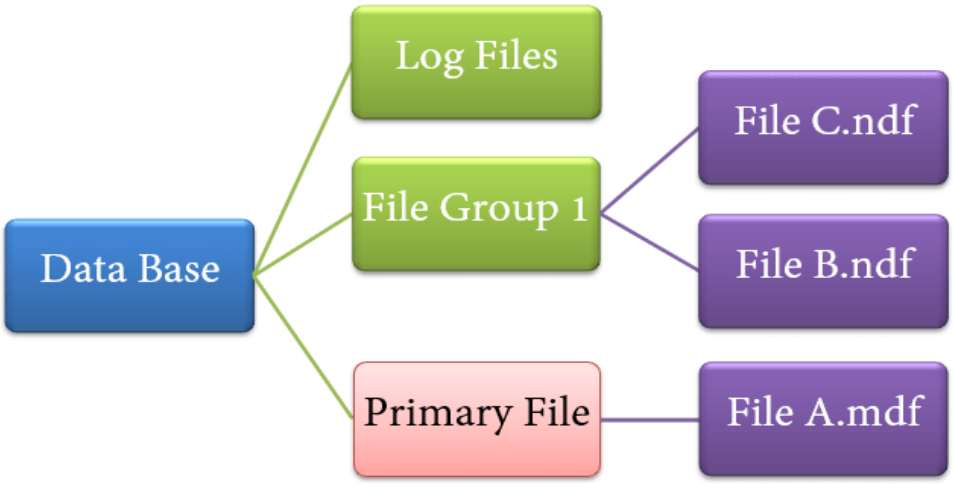

Figure 2 - Schéma de la composition des fichiers d'une base de données

Nous allons nous intéresser à l'organisation de ces types de fichiers.

Organisation des fichiers :

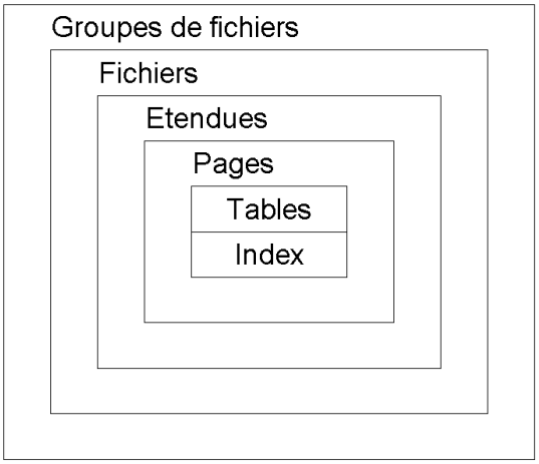

Figure 3 - Représentation de l'organisation du stockage dans un fichier de données

Groupes de fichiers :

Commençons par les groupes de fichiers. SQL Server organise les fichiers en groupes de fichiers, permettant une gestion plus flexible de l'espace disque.

- Le groupe de fichiers principal contient le fichier de données primaire et tous les fichiers secondaires qui ne sont pas placés dans d'autres groupes de fichiers.

- Il est possible de créer des groupes de fichiers définis par l'utilisateur pour regrouper des fichiers de données à des fins d'administration, d'allocation des données et de placement.

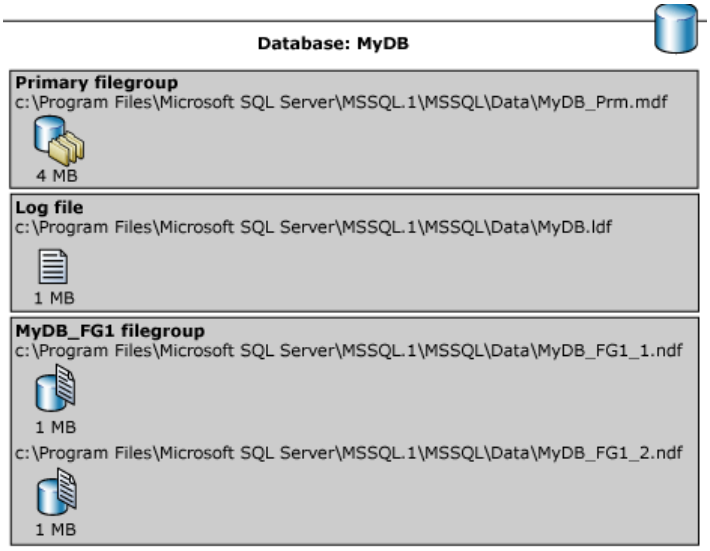

Figure 4 - Illustration de la base de données MyDB

Voici un exemple de base de données avec plusieurs groupes de fichiers. On peut y voir le fichier de données primaire qui fait partie du groupe de fichiers primaire et le groupe de fichiers défini par l'utilisateur possède deux fichiers de données secondaires. Une instruction ALTER DATABASE fait du groupe de fichiers défini par l'utilisateur le groupe par défaut. Cela signifie que lorsque des objets sont créés dans la base de données, sans spécifier le groupe de fichiers auquel ils appartiennent, ces objets sont affectés au groupe de fichiers par défaut.

Tous les fichiers de données sont stockés dans les groupes de fichiers répertoriés dans le tableau suivant.

Groupes de fichiers	Description
Principal	Groupe de fichiers qui contient le fichier primaire. Toutes les tables système sont allouées au groupe de fichiers primaire.
Données optimisées en mémoire	Un groupe de fichiers optimisé en mémoire est basé sur un groupe de fichiers Filestream
Filestream	Permet de stocker des données non structurées, telles que des documents et des images, sur le système de fichiers. On l'utilise si on doit gérer des objets binaires de grande taille (supérieur à 1 mb) tout en maintenant l'intégrité transactionnelle.
Paramétrable	Tout groupe de fichiers créé par l'utilisateur lorsque celui-ci crée la base de données ou lorsqu'il la modifie ultérieurement.

Maintenant que l'on a compris comment les fichiers sont répertoriés dans les différents groupes de fichiers qui existent, on peut s'intéresser à comment elles sont composées.

Pages :

Dans un livre régulier, tout le contenu est écrit sur des pages. Comme pour un livre, SQL Server écrit toutes les lignes de données sur les pages et toutes les pages de données sont de la même taille : 8 Ko. Dans un livre, la plupart des pages contiennent les données (le contenu principal du livre) et certaines pages contiennent des métadonnées sur le contenu (par exemple, la table des matières et l'index). Là encore, SQL Server n'est pas différent : la plupart des pages contiennent des lignes de données réelles stockées par les utilisateurs ; elles sont appelées pages de données et pages texte/image (pour des cas spéciaux). Les pages d'index contiennent des références d'index sur l'emplacement où se trouvent les données. Enfin, il existe des pages système qui stockent différentes métadonnées sur l'organisation des données.

Chaque page commence par un en-tête de 96 octets qui sert à stocker les informations système relatives à la page. Ces informations sont notamment le numéro de page, le type de page, la quantité d'espace disponible sur la page et l'ID de l'unité d'allocation de l'objet auquel appartient la page.

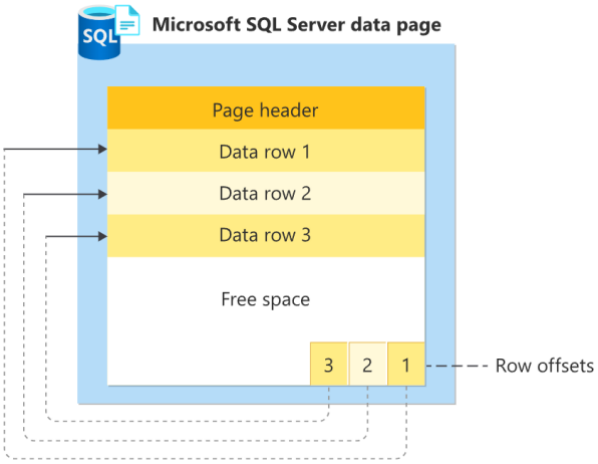

Figure 5 - Représentation d'une page de donnée de SQL Server

Étendues :

Les extensions constituent l'unité de base dans laquelle l'espace est géré. Une extension est constituée de 8 pages, soit 64 Ko. Cela signifie que les bases de données SQL Server ont 16 étendues par mégaoctet.

SQL Server contient deux types d'étendues :

- Les extensions uniformes appartiennent à un objet unique ; les huit pages de l'extension ne peuvent être utilisées que par l'objet propriétaire.
- Les extensions mixtes sont partagées par huit objets au plus. Chacune des huit pages de l'extension peut être la propriété d'un objet différent.

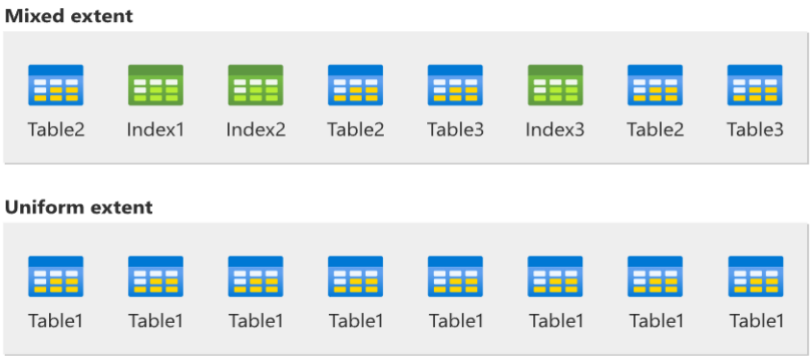

Figure 6 - Représentation de deux types d'étendues

5. Organisation logique

Dans l'architecture de stockage de SQL Server, les concepts tels que les tablespaces, les segments, les extensions et les blocks, couramment utilisés dans d'autres systèmes de gestion de bases de données, sont adaptés et interprétés différemment. Nous allons voir ici la manière dont SQL Server gère ces aspects fondamentaux tels que les tables, les index, les données d'annulation et les données temporaires au sein de son environnement de stockage.

Tables :

Tablespace : SQL Server utilise le terme "Filegroup" plutôt que "Tablespace". Un Filegroup est un ensemble de fichiers physiques où les données d'une table peuvent être stockées.

Segments / Extensions / Blocks : Ces concepts ne sont pas directement utilisés dans SQL Server. Au lieu de cela, les données d'une table sont réparties dans plusieurs datafiles. Un datafile est l'unité de stockage fondamentale, généralement de 8 Ko. Une table est constituée de plusieurs datafiles contenant les enregistrements.

Index :

Tablespace : Comme pour les tables, les index sont également associés à un Filegroup où leurs données peuvent être stockées.

Segments / Extensions / Blocks : De manière similaire aux tables, les données d'un index sont stockées dans des datafiles. Cependant, un index peut également avoir des structures internes spécifiques à son type, telles que des pages d'index intermédiaires pour un index B-arbre.

Données d'Annulation (Undo Data) :

Tablespace : Les données d'annulation dans SQL Server sont gérées par le journal des transactions et ne sont pas associées à un espace de table spécifique.

Segments / Extensions / Blocks : Les données d'annulation sont stockées dans le journal des transactions, qui est une séquence de fichiers appelés logfiles de transactions. Ces fichiers sont organisés en pages, mais l'aspect spécifique des blocs peut être complexe en raison de la nature transactionnelle.

Données Temporaires :

Tablespace : Les données temporaires peuvent être stockées dans des fichiers de base de données temporaires, qui font partie du Filegroup par défaut.

Segments / Extensions / Blocks : Les données temporaires sont également stockées en datafile. Les tables temporaires et les tables de variables de table temporaires utilisent ces pages.

6. Gestion de la concurrence d'accès

6.1 Support des propriétés ACID

Comme tout système de gestion de base de données relationnelle, SQL Server garantit les propriétés ACID pour assurer la fiabilité, l'intégrité et la cohérence des transactions dans un environnement avec plusieurs utilisateurs. Voici comment SQL Server répond à chaque aspect des propriétés ACID :

- **Atomicité :**

Toutes les opérations effectuées au sein d'une transaction sont soit entièrement réalisées, soit entièrement annulées si une erreur survient. Cela est réalisé grâce aux commandes COMMIT et ROLLBACK.

- **Cohérence :**

La cohérence des données est maintenue dans SQL Server grâce à l'utilisation de contraintes telles que les clés primaires et étrangères. Ces contraintes garantissent que la base de données passe d'un état valide à un autre à la fin de chaque transaction. Ainsi, toute opération qui ne respecte pas ces contraintes sera rejetée, assurant ainsi la cohérence des données au sein de la base de données.

- **Isolation** :

SQL Server offre différents niveaux d'isolation, notamment READ COMMITTED, READ UNCOMMITTED, REPEATABLE READ et SERIALIZABLE. Ces niveaux d'isolation permettent de gérer la concurrence des transactions en assurant que chaque transaction se déroule de manière indépendante des autres. Cela garantit que les transactions concurrentes n'interfèrent pas les unes avec les autres, tout en préservant l'intégrité des données.

- **Durabilité** :

La durabilité des transactions dans SQL Server est assurée par les journaux de transactions. Une fois qu'une transaction est confirmée avec succès, grâce à la commande COMMIT, les modifications apportées aux données sont considérées comme permanentes et stockées de manière durable sur le disque. Les journaux de transactions jouent un rôle crucial en enregistrant toutes les modifications apportées, ce qui permet une récupération fiable des données en cas de défaillance du système ou d'interruption.

6.2 Gestion des transactions

Une transaction représente une unité de travail qui, une fois terminée avec succès, valide et intègre définitivement toutes les modifications apportées à la base de données. Si une transaction rencontre des anomalies ou des erreurs, toutes ses modifications sont alors annulées pour maintenir l'intégrité des données.

SQL Server propose plusieurs modes de transaction pour gérer les opérations de manière adaptée :

- **Transaction en mode de validation automatique** :

Chaque instruction individuelle est considérée comme une transaction distincte. Ces transactions nécessitent une validation manuelle à l'aide de la commande "COMMIT" pour que les modifications soient définitivement intégrées.

- **Transactions explicites** :

Ces transactions sont initiées de manière explicite avec l'instruction "BEGIN TRANSACTION" et sont finalisées avec soit "COMMIT" pour valider les changements, soit "ROLLBACK" pour annuler les modifications en cas d'anomalie.

- **Transactions implicites** :

Les transactions implicites sont automatiquement initiées lors de certaines actions, telles que l'activation d'une procédure stockée ou d'un déclencheur. Les modifications effectuées au sein de ces transactions sont validées après chaque instruction, sauf en cas d'erreur où la transaction est automatiquement annulée.

- **Transaction dont l'étendue est définie par lot** :

Lorsqu'une transaction débute sous une session MARS (Multiple Active Result Sets) de SQL Server, cette session permet l'exécution simultanée de plusieurs commandes sans attendre la fin de chacune. La transaction qui en découle est liée au traitement en cours dans cette session. Si cette transaction n'est pas validée ou annulée à la fin du traitement, SQL Server annule automatiquement les modifications non confirmées pour ramener la base de données à son état précédent.

6.3 Gestion des verrous

Dans le contexte multi-utilisateur d'une base de données, un verrou est essentiel pour maintenir l'intégrité des données. Lorsqu'une transaction place un verrou sur une ressource, elle bloque l'accès à cette dernière pour d'autres transactions jusqu'à sa libération, assurant ainsi la cohérence des données. SQL Server gère généralement ces verrous de manière dynamique. Cependant, dans des situations transactionnelles complexes ou exceptionnelles, une intervention peut être nécessaire. Pour cela, des outils tels que les vues de gestion dynamique (DMV), les procédures stockées comme "sp_lock", ou encore le SQL Server Profiler peuvent être utilisés. Dans la plupart des scénarios, ajuster le niveau d'isolation des transactions suffit à résoudre les problèmes liés aux verrous.

Les niveaux de granularité :

SQL Server prend en charge plusieurs niveaux de granularité pour le verrouillage d'objets :

- **Verrouillage de ligne/clé (row ou RID lock/key lock) :** Le verrou est appliqué au niveau de la ligne ou de la clé, particulièrement urilisé dans les index pour le traitement de plages de clés. (Niveau par défaut)

- **Verrouillage de page (page lock)** *:* Le verrou est posé au niveau de la page.

- **Verrouillage de table (table lock)** : Le verrou est établi au niveau de la table entière.

- **Verrouillage d'extent (extent lock)** : Le verrou est appliqué au niveau de l'extent.

- **Verrouillage de base de données (database lock)** : Le verrou est appliqué au niveau de la base de données.

En plus de ces niveaux, d'autres options de verrouillage spécifiques existent, comme le verrouillage HoBT pour les tables en Heap ou les structures d'index B–Tree. Des niveaux de granularité supplémentaires incluent le verrouillage au niveau du fichier, de l'application, de la métadonnée, etc.

Par ailleurs, à partir de SQL Server 2005, des options de granularité spécifiques sont dédiées aux indexes :

- ***ALLOW_ROW_LOCK :*** Autorise le verrouillage au niveau des lignes lors du traitement.
- ***ALLOW_PAGE_LOCK :*** Autorise le verrouillage au niveau des pages lors du traitement d'un index, particulièrement essentiel pour la réorganisation d'un index.

Si ces deux options sont désactivées (OFF), un verrouillage au niveau de la table est utilisé. L'activation simultanée des deux options permet à SQL Server de décider du type de verrouillage à appliquer en fonction du volume de lignes à traiter et des ressources disponibles.

Notion de délai d'attente de verrouillage :

SQL Server propose une directive appelée SET LOCK_TIMEOUT qui permet de définir la durée d'attente en millisecondes pour les opérations de verrouillage. Si une instruction rencontre un verrou et dépasse ce délai sans obtenir le verrou, elle est interrompue, libérant ainsi les ressources.

Par défaut, cette valeur est fixée à -1, ce qui signifie qu'il n'y a pas de limite de temps d'attente et que la transaction en attente demeurera bloquée indéfiniment jusqu'à ce que le verrou soit relâché.

Description des principaux types de verrous :

Les modes de verrouillage traditionnels les plus importants dans SQL Server sont les suivants :

- **Verrous partagés (Shared Locks - S):**

Utilisés pour les opérations de lecture pendant les consultations (SELECT). Ils permettent à plusieurs transactions d'accéder simultanément aux mêmes données en lecture.

- **Verrous exclusifs (Exclusive Locks - X) :**

Utilisés pour les opérations d'écriture lors des modifications de données (INSERT, UPDATE, DELETE). Ils empêchent toute autre session de modifier l'objet concerné pendant le traitement, et ils sont acquis et maintenus jusqu'à la fin de la transaction d'écriture.

- **Verrous de mise-à-jour (Update Locks - U):**

Utilisés lorsqu'une mise à jour est nécessaire. Lorsque l'utilisateur sur SQL Server souhaite mettre à jour une ligne de données, SQL Server pose un verrou de mise-à-jour sur toutes les lignes de la table concernée, puis les parcourt une par une jusqu'à trouver la ligne correspondant à la requête. À ce moment-là, le verrou de mise-à-jour se transforme en un verrou exclusif (X).

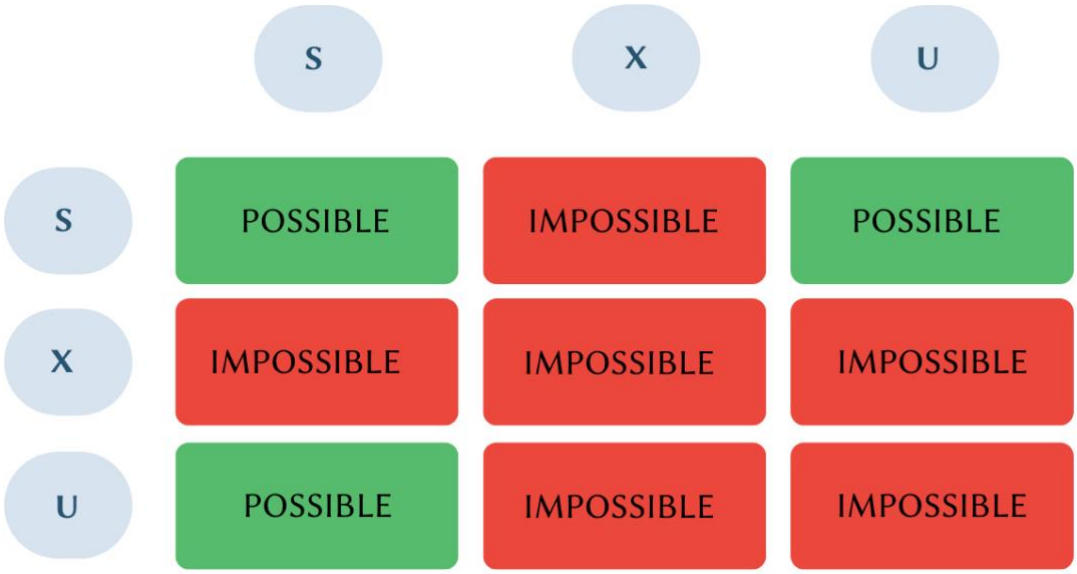

Figure 7 - Matrice de compatibilité des verrous

6.4 Les niveaux d'isolation accessible

Les niveaux d'isolation définissent les règles selon lesquelles les transactions accèdent et interagissent avec les données dans un environnement multi-utilisateur. Chaque niveau utilise des méthodes de verrouillage spécifiques pour prévenir les conflits pouvant être causés par des anomalies potentielles. Ces mécanismes visent à maintenir la cohérence des données malgré les opérations concurrentes.

Lors des opérations de modification telles que INSERT, UPDATE et DELETE, SQL Server applique automatiquement des verrous exclusifs afin de garantir l'intégrité transactionnelle des données. En revanche, pour les requêtes de sélection (SELECT), SQL Server offre des indicateurs de verrouillage qui peuvent être utilisés pour répondre à des exigences spécifiques ou des besoins particuliers.

- **Niveaux d'isolation conformes à la norme SQL :**

- ***READ_UNCOMMITTED*** **:** Autorise l'accès aux données, même si elles sont en cours de modification par d'autres transactions, offrant ainsi la moindre contrainte en matière d'isolation.

- ***READ_COMMITTED :*** Empêche l'accès aux données non validées, reflétant ainsi les modifications déjà confirmées, et c'est le niveau par défaut dans SQL Server.

- ***REPEATABLE_READ :*** Bloque les modifications sur les données lues par une transaction jusqu'à ce que celle-ci soit validée.

- ***SERIALIZABLE :*** Assure une isolation complète en empêchant tout accès concurrent aux données, prévenant ainsi les anomalies transactionnelles comme les lectures fantômes.

	Mise à jour perdue	Lectures impropres	Lectures non reproductibles	Tuples fantômes
Read uncommited	IMPOSSIBLE	POSSIBLE	POSSIBLE	POSSIBLE
Read commited	IMPOSSIBLE	IMPOSSIBLE	POSSIBLE	POSSIBLE
Repeatable read	IMPOSSIBLE	IMPOSSIBLE	IMPOSSIBLE	POSSIBLE
Serialisable	IMPOSSIBLE	IMPOSSIBLE	IMPOSSIBLE	IMPOSSIBLE

Figure 8 - Tableau des niveaux d'isolation et anomalies

Le choix du niveau d'isolation repose sur un équilibre entre performance et intégrité des données. Bien que SERIALIZABLE offre une protection maximale, il peut ralentir les performances. À l'opposé, READ UNCOMMITTED est moins contraignant mais peut conduire à des incohérences. READ_COMMITTED offre un bon compromis pour la plupart des scénarios, mais en présence de nombreux deadlocks, l'utilisation de READ COMMITTED SNAPSHOT pourrait être considérée, tout en restant vigilant aux problèmes potentiels de conception ou d'optimisation des requêtes.

- **Niveaux d'isolation basés sur l'instantané (Snapshot) :**

À partir de SQL Server 2005, les niveaux SNAPSHOT et READ COMMITTED SNAPSHOT ont été introduits pour faciliter les accès concurrents. Ces niveaux utilisent le versionnage des lignes, conservant les versions antérieures des données dans un "version store" de tempDB. Ainsi, les lectures peuvent se faire sans blocage, en se basant sur la dernière version non modifiée des données.

1. SNAPSHOT : Permet des lectures sans interférence en consultant la version antérieure des données stockée dans la version store de tempDB.

2. READ COMMITTED SNAPSHOT : Une extension de READ COMMITTED, combinant ses avantages avec ceux de SNAPSHOT, mais sans détection de conflits de mise à jour.

7. Gestion des transactions distribuées

7.1 Architecture des transactions distribuées

La gestion des transactions distribuées dans SQL Server constitue un élément crucial pour assurer la cohérence des données dans des environnements complexes. Avec le Distributed Transaction Coordinator (DTC) de Microsoft SQL Server, la coordination efficace des transactions distribuées est facilitée, garantissant la fiabilité des opérations sur des instances variées et des configurations diverses.

- **Participants de la Transaction Distribuée :**

Les acteurs impliqués dans une transaction distribuée comprennent les instances de SQL Server, les bases de données agissant comme des gestionnaires de ressources lorsqu'elles sont configurées avec DTC_SUPPORT = PER_DB au sein d'un groupe de disponibilité, le service de Coordinateur de Transactions Distribuées (DTC), et d'autres sources de données.

- **Inscription et Coordination :**

Chaque instance de SQL Server agit en tant que gestionnaire de ressources en s'inscrivant dans la transaction DTC avec un identificateur unique (RMID). Le DTC assure la coordination de la transaction, gérant les phases d'inscription, de préparation, et de validation entre les différents gestionnaires de ressources.

- **Promotion des Transactions :**

Les transactions entre bases de données au sein d'une même instance de SQL Server sont élevées au niveau du DTC lorsque les bases de données sont configurées avec DTC_SUPPORT

= PER_DB. Cette promotion vers le DTC est essentielle pour traiter ces transactions comme des opérations distribuées.

- **Validation en Deux Phases :**

Le processus de validation en deux phases est essentiel pour garantir l'intégrité des transactions distribuées. Le DTC initie la phase de préparation, envoyant une commande à tous les gestionnaires de ressources qui confirment ou signalent un échec. En cas de succès, le DTC envoie une commande de validation, sinon, une commande de restauration est émise.

- **Procédure de Gestion des Transactions Distribuées :**

Lorsqu'une transaction distribuée est en cours, chaque instance de SQL Server s'inscrit dans la transaction DTC. Le client effectue les opérations sous cette transaction, puis émet une commande de validation ou d'abandon. Le DTC coordonne les phases de préparation et de validation avec les gestionnaires de ressources, renvoyant une notification de réussite à l'application en cas de succès, ou émettant une commande de restauration en cas d'échec.

- **Effets de la Configuration d'un Groupe de Disponibilité :**

Dans un groupe de disponibilité configuré avec DTC_SUPPORT = PER_DB, chaque base de données agit comme un gestionnaire de ressources distinct avec un RMID unique. Cette configuration permet une participation efficace à des transactions distribuées, même lors du déplacement d'une base de données vers une autre instance de SQL Server.

7.2 Positionnement par rapport aux 12 règles de Date

Voici comment SQL Server se positionne par rapport à ces règles clés.

1. **Autonomie locale, gestion locale des données, contrôle local des contraintes et de la sécurité :**

Une base de données partiellement autonome dans SQL Server cherche à fonctionner de manière indépendante en étant isolée des autres entités de l'instance. Elle conserve et gère ses métadonnées essentielles directement à l'intérieur de sa propre structure, réduisant ainsi sa dépendance par rapport aux métadonnées stockées dans la base de données MASTER. Cette approche lui permet de maintenir un certain degré d'autonomie locale et de contrôle spécifique sur ses fonctionnalités et ses données, favorisant ainsi une gestion autonome et indépendante.

2. **Indépendance vis-à-vis d'un site central :**

SQL Server, lorsqu'il est utilisé dans un environnement distribué, peut être configuré pour fonctionner en mode distribué, évitant ainsi un point central unique pour l'accès aux données

3. **Fonctionnement continu :**

Avec des stratégies de haute disponibilité comme Always On Availability Groups, SQL Server permet un fonctionnement continu, même lors de mises à jour ou de changements de version.

4. **Indépendance vis-à-vis de la localisation :**

Les fonctionnalités de SQL Server pour la réplication et la distribution de données permettent à l'utilisateur de manipuler les données sans être conscient de leur localisation physique.

5. **Indépendance vis-à-vis de la fragmentation :**

SQL Server peut gérer le partitionnement des données via des fonctionnalités telles que le partitionnement de table, permettant aux utilisateurs de travailler avec des données sans en percevoir la fragmentation.

6. **Indépendance vis-à-vis de la duplication :**

Les fonctionnalités de réplication de SQL Server permettent de dupliquer les données sur plusieurs sites de manière transparente pour les utilisateurs.

7. **Traitement réparti des requêtes :**

SQL Server optimise les requêtes en utilisant des stratégies globales et locales pour améliorer les performances, bien que cette optimisation puisse dépendre de la configuration et des indices.

8. **Gestion répartie des transactions :**

SQL Server prend en charge les transactions distribuées à l'aide de mécanismes comme le commit à deux phases pour assurer l'atomicité globale des transactions indépendamment des sites.

9. **Indépendance vis-à-vis du matériel :**

SQL Server est conçu pour fonctionner sur une variété de configurations matérielles, offrant ainsi une certaine indépendance vis-à-vis du matériel. Cependant, les performances peuvent varier en fonction des ressources matérielles disponibles.

10. **Indépendance vis-à-vis de l'OS :**

SQL Server est disponible pour différentes plateformes et versions d'OS, offrant une certaine indépendance vis-à-vis de l'OS.

11. **Indépendance vis-à-vis du réseau :**

SQL Server peut fonctionner sur différents types de réseaux, mais les performances peuvent être influencées par la qualité et la latence du réseau. Il utilise des protocoles standards pour la communication, mais une bonne infrastructure réseau est souvent nécessaire pour des performances optimales.

12. **Indépendance vis-à-vis du SGBD :**

Bien que SQL Server soit un système de gestion de base de données spécifique de Microsoft, il peut interagir avec d'autres SGBD via des fonctionnalités telles que la connectivité externe ou

l'intégration avec d'autres plateformes. Cependant, il est souvent plus compatible et performant lorsqu'il est utilisé avec des technologies de la même famille (Microsoft).

7.3 Méthode pour garantir l'atomicité, la cohérence et la durabilité

Pour garantir les propriétés ACID dans SQL Server, plusieurs mécanismes sont mis en place :

- **Utilisation du Deux-Phase Commit (2PC)** :

La méthode traditionnelle pour assurer la cohérence dans les transactions distribuées est le protocole de commit en deux phases (2PC).
La première phase consiste en une préparation, où chaque participant notifie s'il est prêt à effectuer la transaction.
La seconde phase est le commit ou le rollback, où tous les participants exécutent la transaction ou annulent en fonction des réponses reçues.

- **Utilisation de Bibliothèques et de Frameworks de Transactions Distribuées :**

Des outils et des frameworks comme Java Transaction API (JTA), Microsoft Distributed Transaction Coordinator (DTC), et d'autres fournissent des fonctionnalités pour gérer les transactions distribuées.
Ces frameworks implémentent souvent des mécanismes avancés pour maintenir les propriétés ACID.

- **Points de contrôle (Checkpoints)** :

SQL Server utilise des points de contrôle pour enregistrer les informations nécessaires pour restaurer les bases de données à un état cohérent après une panne. Les checkpoints réduisent le temps de récupération nécessaire après un incident.

- **Journalisation et Rétention des Journaux :**

Maintenez des journaux détaillés de toutes les opérations effectuées au cours de la transaction. La journalisation permet de récupérer l'état précédent en cas d'échec, assurant ainsi la durabilité.

7.4 Traitement des pannes dans un environnement distribué

La prise en compte des pannes est cruciale pour assurer la continuité des opérations. Ainsi, nous allons voir les mécanismes et les stratégies mis en place par SQL Server pour traiter efficacement les pannes, pour garantir la disponibilité et minimiser les pertes de données en cas d'incidents.

- **Redondance et réplication :**

SQL Server propose des fonctionnalités de réplication pour maintenir des copies redondantes des données sur plusieurs serveurs. Cela permet une meilleure disponibilité des données et facilite la reprise après une panne en utilisant les copies redondantes.

- **Clustering et haute disponibilité (always on) :**

L'utilisation de clusters dans SQL Server permet de maintenir la disponibilité en cas de panne matérielle. Les clusters permettent la transition transparente vers un nœud de secours si un nœud principal tombe en panne.

- **Points de contrôle (Checkpoints) :**

SQL Server utilise des checkpoints pour enregistrer les modifications non enregistrées dans les fichiers de données et pour maintenir des journaux de transactions cohérents. Cela aide à minimiser la perte de données en cas de panne.

8. Gestion de la reprise sur panne

8.1 Les techniques d'annulation

La gestion de reprise sur panne dans SQL Server repose sur l'utilisation de techniques d'annulation, qui sont essentielles pour restaurer la cohérence des données après une panne. Ces techniques s'appuient sur la gestion des transactions et la journalisation.

- L'algorithme de **ROLLBACK** est utilisé pour annuler les transactions non confirmées en cas de panne. Il annule toutes les modifications depuis le début de la transaction.

- Les points de sauvegarde permettent de définir des points à l'intérieur d'une transaction pour effectuer un rollback, annulant les modifications depuis le dernier point de sauvegarde.

- La commande **SET XACT_ABORT ON** active l'annulation automatique de la transaction en cas d'erreur, assurant une gestion fiable des transactions en cas de problème.

- Le bloc **TRY...CATCH** est utilisé pour gérer les erreurs et effectuer un rollback en cas d'échec de la transaction, assurant ainsi l'intégrité des données même en cas d'erreur.

Architecture des techniques d'annulation :

L'architecture des techniques d'annulation dans SQL Server repose sur un système sophistiqué de gestion des transactions, de journaux et de mémoires tampons.

Architecture Physique du Journal des Transactions :

Le journal des transactions en SQL Server enregistre toutes les transactions et modifications de données dans une base de données, avec chaque enregistrement associé à un numéro de séquence de journal (LSN - Log Sequence Number). Les types d'opérations enregistrées incluent le début et la fin de transactions, les mises à jour de données, la création/suppression de tables ou d'index, et les opérations de restauration. Les fichiers journaux virtuels (VLF -

Virtual Log File) sont des unités de journalisation qui stockent les transactions, gérées automatiquement par SQL Server en fonction de la croissance du fichier journal. Il est crucial de gérer la distribution des VLF pour éviter des problèmes de performance.
Trop de VLF peuvent causer des problèmes, tels que des temps de récupération prolongés, des erreurs d'attachement de bases de données, et des latences de réplication. Il est recommandé de créer des fichiers journaux proches de la taille finale requise avec des incréments appropriés pour une distribution optimale des VLF. La troncation du journal est essentielle pour éviter le remplissage complet du journal, et elle se produit automatiquement après un point de contrôle ou une sauvegarde du journal.

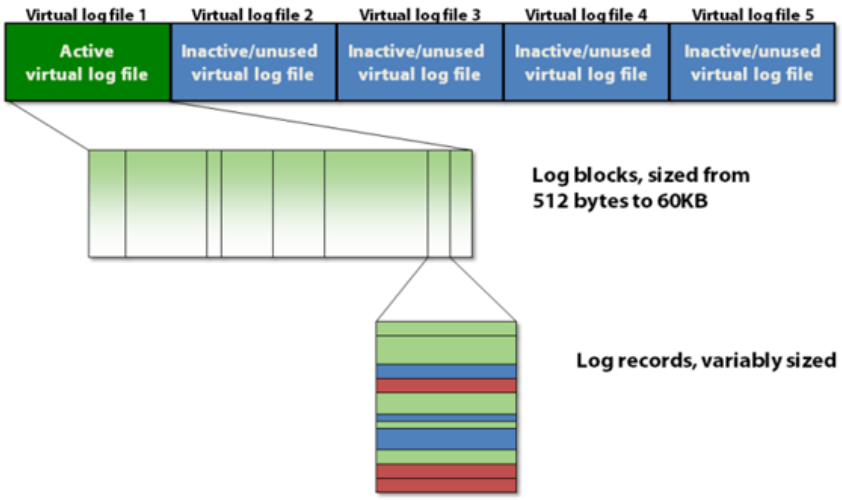

Figure 9 - Architecture du journal des transactions

Journal des transactions à écriture anticipée (WAL) :

Le journal WAL enregistre toutes les modifications apportées aux données avant qu'elles ne soient physiquement écrites sur le disque. SQL Server utilise un cache de mémoire tampon pour gérer les pages de données et ne réécrit pas immédiatement les pages modifiées sur le disque. C'est sur le Transaction Log Buffer.

Chaque modification logique génère un enregistrement de journal des transactions qui doit être écrit sur le disque avant que la page modifiée ne soit supprimée du cache et écrite sur le disque.

Le processus de point de contrôle analyse régulièrement le cache et écrit les pages modifiées sur le disque, créant ainsi un point de récupération pour les données.

Mémoires tampons :

Les mémoires tampons sont utilisées pour stocker temporairement les pages de données provenant des fichiers de base de données.

Buffer Pool : C'est une zone de la mémoire vive (RAM) réservée pour stocker ces mémoires tampons. Le gestionnaire de mémoire tampon (Buffer Manager) est responsable de la gestion du Buffer Pool.

Lorsqu'une modification est apportée à une page de données, elle est d'abord modifiée dans le cache mémoire (mémoire tampon) avant d'être écrite sur le disque. Cela permet d'améliorer les performances en minimisant l'accès au disque.

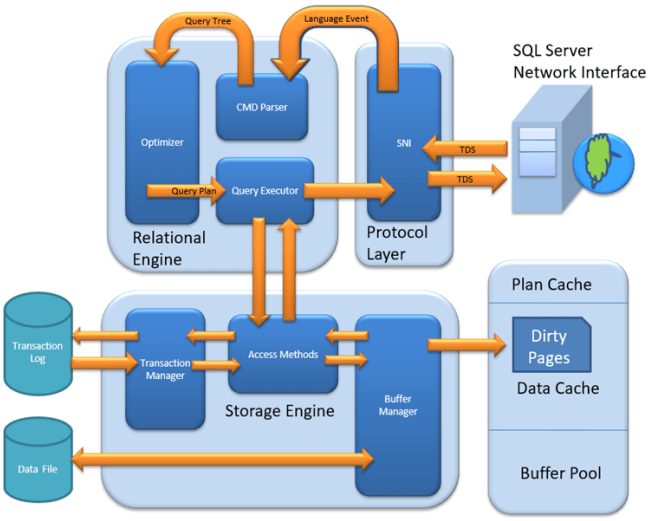

Figure 10 - Architecture mémoire tampons et journaux

8.2 La journalisation

La journalisation dans SQL Server est essentielle pour la gestion de reprise sur panne. Elle se compose principalement du journal des transactions et du journal des erreurs. Le journal des transactions enregistre les modifications de données, tandis que le journal des erreurs capture les événements liés au serveur SQL Server.
Les informations contenues dans le journal des erreurs peuvent être cruciales pour le dépannage, la surveillance et la maintenance du serveur SQL Server.

- **Journal de transactions :**

SQL Server utilise un journal de transactions pour enregistrer les modifications apportées aux données. Il s'agit d'une séquence d'enregistrements, chacun décrivant une opération telle qu'une mise à jour, une validation ou une suppression de transaction.

Il garantit l'intégrité des données et permet la récupération après une panne. Pour activer la journalisation, une base de données doit être spécifiée comme "journalisée". Cela peut être effectué à l'aide de l'instruction ALTER DATABASE.

Avant que les modifications ne soient écrites dans le fichier de journal, elles sont d'abord stockées dans un tampon spécifique : le Transaction Log Buffer. Les modifications enregistrées dans le fichier de journal sont ensuite propagées vers les fichiers de données, assurant ainsi la durabilité des transactions.

- **Sauvegardes des journaux de transactions :**

Il est essentiel de sauvegarder régulièrement les journaux de transactions, généralement toutes les 15 à 30 minutes, pour permettre la récupération des données. Cela permet de libérer de l'espace dans le journal après confirmation que les transactions correspondantes ont été correctement enregistrées dans la base de données.

Il est recommandé de commencer par une sauvegarde complète de la base de données avant de sauvegarder le journal des transactions.

Les sauvegardes régulières permettent également la troncation du journal des transactions. Les sauvegardes du journal des transactions sont essentielles pour la récupération des données jusqu'au point dans le temps de la dernière sauvegarde.

Les autorisations BACKUP DATABASE et BACKUP LOG sont généralement accordées aux membres des rôles sysadmin, db_owner et db_backupoperator.

Pour sauvegarder un journal de transactions :

- **Utilisation de SQL Server Management Studio (SSMS)** : interface graphique permettant de sélectionner la base de données dont on veut sauvegarder le journal.
- **Utilisation de Transact-SQL** : L'instruction BACKUP LOG permet de sauvegarder le journal des transactions.
- **Utilisation de PowerShell** : En utilisant la commande Backup-SqlDatabase pour sauvegarder le journal des transactions.

- **Restauration des sauvegardes de journaux :**

Les enregistrements de journal pour les modifications non écrites sur le disque avant un arrêt sont restaurés lors de la récupération. Toutes les transactions incomplètes sont également restaurées.

SQL Server génère des points de contrôle automatiques pour réduire la durée de récupération de la base de données. On peut aussi le faire manuellement avec CHECKPOINT dans une transaction.

L'option de configuration "intervalle de récupération" peut être utilisée pour définir la durée maximale d'une opération de récupération.

- **Audit SQL Server :**

SQL Server offre également des fonctionnalités d'audit pour suivre des activités spécifiques. En résumé, la journalisation est cruciale pour garantir la fiabilité des données, la récupération après panne et la surveillance du serveur SQL Server.

8.3 Les sauvegardes

La sauvegarde dans SQL Server est cruciale pour la préservation des données et la récupération après une panne. Une stratégie de sauvegarde régulière doit être définie, adaptée aux besoins de l'entreprise.

Types de sauvegardes :

- **Sauvegardes complètes :** Copie l'intégralité de la base de données, y compris les fichiers de données et les journaux de transactions. Elle est utilisée comme point de départ pour les sauvegardes différentielles et de journal des transactions. Elle crée un fichier de sauvegarde (**.bak**).

- **Sauvegardes différentielles :** Copie uniquement les modifications depuis la dernière sauvegarde complète. Elle est plus rapide que la sauvegarde complète et permet de réduire le temps nécessaire pour les restaurations.

- **Sauvegardes de journaux de transactions :** Copie les transactions enregistrées dans le journal depuis la dernière sauvegarde des journaux.

- **Sauvegarde partielle / de fichiers ou de groupes de fichiers (File or Filegroup Backup) :** Il est possible de sauvegarder des fichiers ou des groupes de fichiers spécifiques au lieu de sauvegarder l'ensemble de la base de données. Cette sauvegarde est souvent utilisée dans les bases de données volumineuses.

Il faut définir une stratégie de sauvegarde régulière pour la base de données.

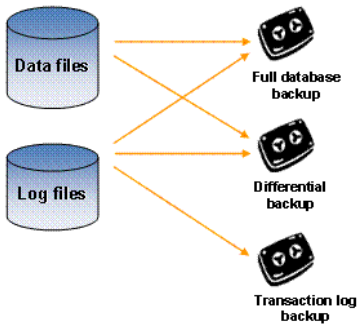

Figure 11 - Types de sauvegarde et les fichiers sur lesquels ils interviennent

Algorithmes de Récupération :

Algorithme REDO :

Cet algorithme est utilisé pour appliquer les modifications enregistrées dans les journaux de transactions après un point de contrôle. Il s'assure que toutes les modifications qui ont été confirmées sont appliquées pour ramener la base de données à un état cohérent.

Algorithme UNDO :

Cet algorithme est utilisé pour annuler les transactions non confirmées en appliquant les opérations inverses des modifications enregistrées dans les journaux de transaction.
Les commandes de restauration sont utilisées pour appliquer la récupération des journaux.

- **Restauration d'une sauvegarde complète :** En cas de panne majeure, une sauvegarde complète peut être restaurée pour ramener la base de données à son état au moment de la sauvegarde.
- **Restauration différentielle ou des journaux de transactions**
- **Point-in-Time Recovery :** SQL Server permet une récupération jusqu'à un point spécifique dans le temps en utilisant les sauvegardes de journaux de transactions.

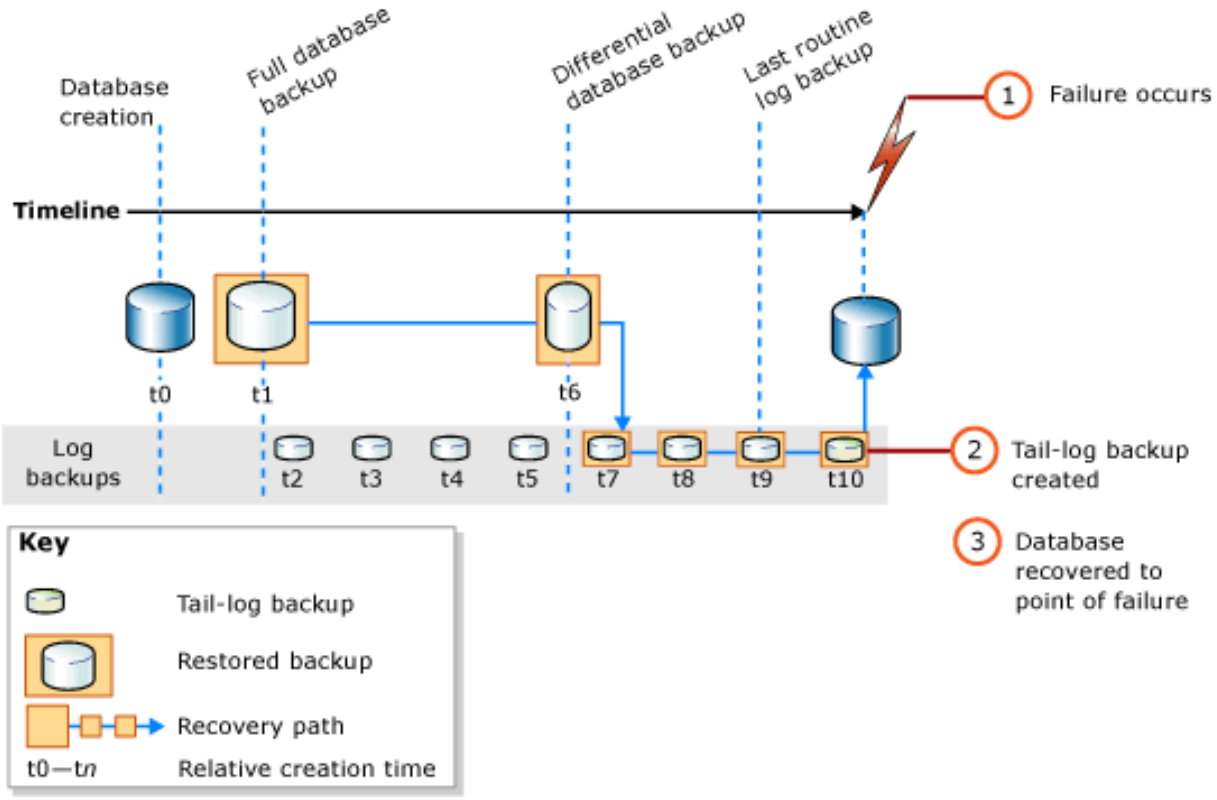

Figure 12 - Schéma restauration complète de base de données avec les différents types de sauvegardes

Stratégies de sauvegarde :

Planification des sauvegardes :

Définir un plan de sauvegarde régulier, adapté aux besoins de l'entreprise ainsi qu'à la criticité des données. Cela peut inclure des sauvegardes complètes régulières et des sauvegardes différentielles ou de journaux plus fréquents. SQL Server Agent peut être utilisé pour planifier automatiquement les sauvegardes à des intervalles définis.

Stockage des sauvegardes :

Sauvegarder les données sur un support externe sécurisé pour éviter la perte de données en cas de défaillance matérielle. En plus du stockage local, SQL Server prend en charge la sauvegarde et la restauration à partir du Stockage Blob Azure.

Il est essentiel de planifier les sauvegardes, de stocker les sauvegardes de manière sécurisée, et de les tester régulièrement en effectuant des restaurations sur des environnements de test. Cependant, avant de mettre en place une stratégie de sauvegarde, il est recommandé d'estimer la taille des sauvegardes en utilisant *sp_spaceused*.

8.4 Gestion de la reprise à chaud

La gestion de reprise de pannes à chaud dans SQL Server concerne la capacité à restaurer la base de données et à assurer la continuité des opérations même lorsque des utilisateurs accèdent à la base de données pendant la phase de récupération.

Cela implique des mécanismes tels que la journalisation, les sauvegardes et la récupération à base de journaux. SQL Server propose aussi plusieurs fonctionnalités et solutions pour mettre en œuvre une reprise à chaud, dont les plus importantes sont AlwaysOn Availability Groups et la journalisation des transactions.

AlwaysOn Availability Groups :

AlwaysOn Availability Groups est une fonctionnalité de haute disponibilité introduite dans SQL Server. Elle permet de créer des groupes de disponibilité qui regroupent plusieurs bases de données. Chaque groupe de disponibilité peut avoir une copie primaire et une ou plusieurs copies secondaires. En cas de panne du serveur primaire, l'une des copies secondaires peut être immédiatement activée en tant que serveur primaire, assurant ainsi la continuité des opérations sans interruption.

La transition d'une copie secondaire à une copie primaire se fait de manière automatisée à l'aide du **gestionnaire de cluster Windows** et de **SQL Server AlwaysOn**.

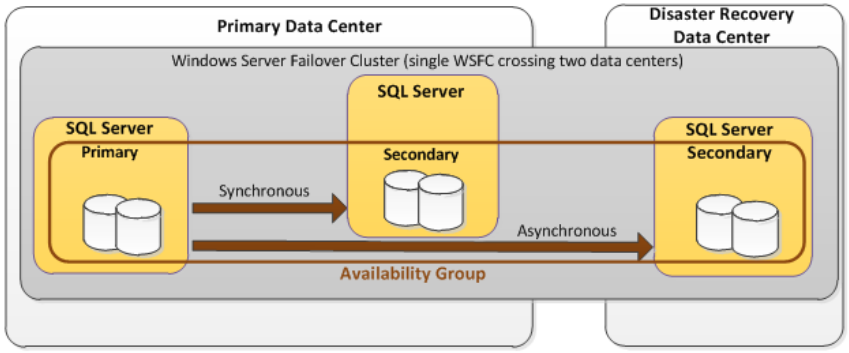

Figure 13 - Schéma Availability Group

Mirroring de base de données :

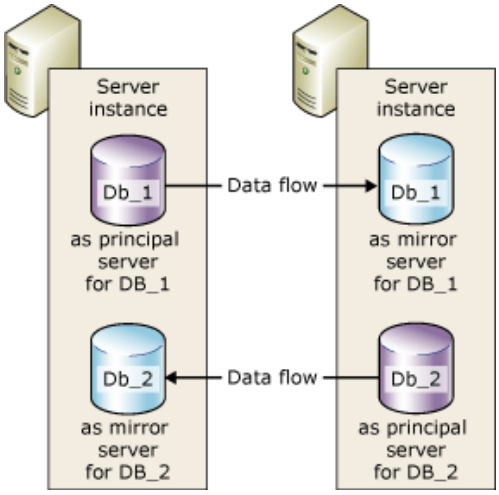

Figure 14 - Schéma Mirroring de base de données

SQL Server Database Mirroring est une autre option qui permet de maintenir une copie synchronisée d'une base de données sur un autre serveur. Lorsqu'une panne survient sur le serveur principal, le miroir peut être activé pour prendre en charge les opérations.

Journalisation des transactions :

La journalisation des transactions est une technique où les transactions sont régulièrement sauvegardées et restaurées sur un serveur distant. Cette approche permet de maintenir une copie à jour de la base de données sur le serveur distant. En cas de panne du serveur principal, le serveur distant peut être activé pour prendre en charge les opérations.

Cluster de basculement :

SQL Server peut également être configuré en tant qu'instance de cluster de basculement. Dans ce scénario, plusieurs serveurs sont configurés pour partager le même stockage et l'un d'eux est actif à la fois. En cas de panne du serveur actif, l'instance de cluster bascule vers un autre nœud, offrant ainsi une reprise à chaud.

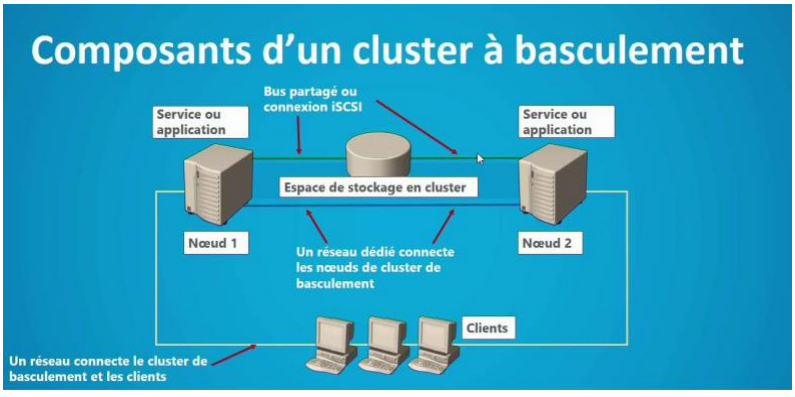

Figure 15 - Schéma des composants d'un cluster à basculement

Sauvegardes régulières :

Mettre en place une stratégie de sauvegarde régulière, comprenant :

- Sauvegarde complète quotidienne
- Sauvegarde différentielle régulière
- Sauvegarde des journaux de transactions fréquente

Récupération à chaud :

Supposons qu'une panne survient, et que la base de données doit être restaurée après une défaillance du système.

- Restauration de la sauvegarde complète
- Restauration de la sauvegarde différentielle
- Restauration des journaux de transactions

Une fois toutes les sauvegardes restaurées et la récupération à base de journaux terminée, la base de données est prête à être utilisée normalement.

Rôle du journal de transactions pendant la reprise à chaud :

Le journal de transactions enregistre les modifications effectuées pendant la phase de récupération à chaud.

Rollback ou REDO : Les transactions en attente d'annulation ou de validation sont gérées par la récupération à base de journaux, qui peut effectuer des opérations de rollback ou de refaire selon les besoins.

La gestion de reprise de pannes à chaud dans SQL Server, en utilisant la journalisation, les sauvegardes et la récupération à base de journaux, assure une disponibilité continue des données même pendant la phase de récupération. Elle permet de minimiser les temps d'arrêt et de maintenir l'intégrité des transactions, offrant ainsi une solution robuste pour les environnements nécessitant une haute disponibilité.

8.5 Gestion de la reprise à froid

La gestion de reprise de pannes à froid vise à restaurer une base de données après une panne lorsque le système est hors service à partir de sauvegardes sur un serveur de secours après la défaillance du serveur principal.

Contrairement à la reprise à chaud, où la transition vers un serveur de secours se fait sans interruption, la reprise à froid nécessite généralement un temps d'arrêt pendant lequel les données sont restaurées.

Sauvegardes régulières :

Tout comme dans la reprise à chaud, une stratégie de sauvegarde régulière est essentielle pour la gestion de reprise de pannes à froid comprenant sauvegarde complète quotidienne, sauvegarde différentielle régulière ainsi que des sauvegardes de journaux fréquentes.

Arrêt du service SQL Server :

Supposons qu'une panne majeure survienne, nécessitant la récupération à froid.

Arrêter le service SQL Server pour garantir qu'aucune transaction n'est en cours pendant la phase de récupération à froid.

Copie des fichiers de sauvegarde sur le nouvel emplacement :

Imaginons que la base de données nécessite une récupération à froid sur un nouveau serveur ou un nouvel emplacement. Il faut copier les fichiers de sauvegarde sur le nouvel emplacement.

Restauration des sauvegardes à froid :

Imaginons que le service SQL Server a été arrêté, les fichiers ont été copiés sur un nouvel emplacement, et maintenant, la base de données doit être restaurée. Ces opérations sont généralement effectuées à l'aide de commandes SQL telles que RESTORE DATABASE et RESTORE LOG.

Redémarrage du service SQL Server :

Une fois la base de données restaurée à froid, le service SQL Server peut être redémarré pour permettre aux utilisateurs d’accéder à la base de données.

La gestion de reprise de pannes à froid dans SQL Server implique donc l'arrêt du service, la copie des fichiers de sauvegarde vers un nouvel emplacement, la restauration des sauvegardes, le redémarrage du service et enfin la reprise des opérations normales. Cette approche assure une récupération après une panne, même lorsque le système est hors service, tout en minimisant la perte de données

9. Techniques d’indexation

Un index est une structure sur disque associée à une table ou une vue qui accélère l'extraction des lignes de la table ou de la vue. Il contient des clés créées à partir d'une ou plusieurs colonnes de la table ou de la vue. Ces clés sont stockées dans une structure (B-tree) qui permet à SQL Server de trouver rapidement et efficacement la ou les lignes associées aux valeurs de clé.

Il existe plusieurs types d’indexation dans SQL server, voici un tableau avec celles qui me paraissent les plus importantes et quelques exemples :

Type d'index	Description
Hash	Avec un index de hachage, les données sont accédées via une table de hachage en mémoire. Les index de hachage consomment une quantité fixe de mémoire, en fonction du nombre de compartiments.
Index non-cluster à mémoire optimisée	Pour les index non-cluster optimisés en mémoire, la consommation de mémoire est une fonction du nombre de lignes et de la taille des colonnes de clés d'index
Cluster	Un index cluster trie et stocke les lignes de données de la table ou de la vue en fonction de la clé d'index cluster. L'index cluster est mis en œuvre sous la forme d'une structure d'index arborescente binaire permettant la récupération rapide des lignes d'après leurs valeurs clés de l'index cluster. **Exemple** : Supposons une table "Utilisateurs" avec un index clusterisé sur la colonne "ID_Utilisateur". Les lignes de la table seraient organisées physiquement selon l'ordre croissant des ID_Utilisateur.
Non-cluster	Les index non-cluster peuvent être définis dans une table ou dans une vue dotée d'un index cluster ou d'un segment de mémoire. Chaque ligne d'un index non-cluster contient la valeur clé non cluster ainsi qu'un localisateur de ligne. Le localisateur pointe vers la ligne de données dans l'index cluster ou dans le segment doté de la valeur clé. Les lignes de l'index sont stockées selon l'ordre des valeurs clés de l'index. L'ordre spécifique des lignes de données n'est garanti que si un index cluster est créé sur la table. **Exemple** : Une table "Commandes" avec un index non clusterisé sur la colonne "Num_Commande". Les lignes restent physiquement dans l'ordre d'insertion, mais un index séparé est créé pour accélérer les recherches basées sur les numéros de commande.
columnstore	Un index columnstore en mémoire stocke et gère des données à l'aide du stockage des données en colonnes et du traitement des requêtes en colonnes. Les index columnstore fonctionnent bien pour les charges de travail de stockage de données qui effectuent principalement des chargements en masse et des requêtes en lecture seule. Utilisez l'index columnstore pour atteindre des performances des requêtes pouvant être multipliées par 10 par rapport au stockage orienté lignes traditionnel, et une compression de données multipliée par 7 par rapport à la taille des données non compressées. **Exemple** : Une table "Ventes" avec un index columnstore sur les colonnes "Montant_Vente" et "Date_Vente" pour optimiser les requêtes d'agrégation sur ces colonnes.
Index à colonnes incluses	Index non-cluster qui est étendu pour inclure des colonnes non-clés en plus des colonnes clés.

Index sur les colonnes calculées	Index sur une colonne dérivée de la valeur d'une ou de plusieurs autres colonnes, ou certaines entrées déterministes.
Filtré	Index non-cluster optimisé, convenant tout particulièrement aux requêtes qui effectuent des sélections dans un sous-ensemble précis de données. Il utilise un prédicat de filtre pour indexer une partie des lignes de la table. Un index filtré bien conçu peut améliorer les performances des requêtes, réduire les coûts de maintenance des index et réduire les coûts de stockage des index par rapport aux index de table entière. exemple : Une table "Produits" avec un index filtré sur les produits "en stock" seulement, pour accélérer les requêtes qui portent sur les produits disponibles.
Spatial	Un index spatial permet d'effectuer plus efficacement certaines opérations sur des objets spatiaux (*données spatiales*) dans une colonne du type de données géométrie. L'index spatial réduit le nombre d'objets sur lesquels des opérations spatiales relativement coûteuses doivent être appliquées. **Exemple** : Une table "Lieux" avec un index spatial sur la colonne de géolocalisation pour accélérer les recherches de lieux dans une zone spécifique.

Création d'index par rapport aux contraintes :

SQL Server crée automatiquement des index lorsque les contraintes PRIMARY KEY et UNIQUE sont définies sur les colonnes de table. Par exemple, lorsque vous créez une table avec une contrainte UNIQUE, Moteur de base de données crée automatiquement un index non-cluster. Si vous configurez une CLÉ PRIMAIRE, Moteur de base de données crée automatiquement un index cluster, sauf si un index cluster existe déjà.

Segments (tables sans index cluster) :

Le seul moment où les lignes de données d'une table sont stockées dans l'ordre de tri se produit lorsque la table contient un index cluster. Lorsqu'une table possède un index cluster, elle est appelée table cluster. En l'absence de cet index, ses lignes de données sont stockées dans une structure désordonnée nommée segment.

Un segment est une table sans index cluster. Un ou plusieurs index non-cluster peuvent être créés sur des tables stockées comme segment. Les données sont stockées dans le segment sans spécifier d'ordre. Généralement, les données sont initialement stockées dans l'ordre dans lequel les lignes sont insérées dans la table, mais le moteur de base de données peut déplacer les données dans le tas pour stocker efficacement les lignes ; l'ordre des données ne peut donc pas être prédit. Pour garantir l'ordre des lignes retournées à partir d'un segment, vous devez utiliser la clause ORDER BY.

Les segments de mémoire peuvent être utilisés en tant que tables de mise en lots pour des opérations d'insertion volumineuses et non ordonnées. Étant donné que les données sont insérées sans appliquer un ordre strict, l'opération d'insertion est généralement plus rapide que l'insertion équivalente dans un index cluster.

10. Optimisation de requêtes

Explorons désormais les différentes approches d'optimisation des requêtes dans SQL Server. Chaque méthode joue un rôle crucial dans l'amélioration des performances des requêtes au sein de la base de données. Un suivi attentif des performances, grâce à des outils spécialisés, complète cette démarche en permettant d'identifier et de rectifier les zones nécessitant des améliorations.

- ## Optimisation Manuelle :

Définition : L'optimisation manuelle des requêtes implique que les développeurs ou les administrateurs de base de données ajustent les requêtes SQL pour améliorer leurs performances.

Exemple : Réécrire une requête en utilisant des jointures plus efficaces, spécifier des indexes, ou ajuster l'ordre d'exécution des opérations.

- ## Utilisation d'Index :

Définition : La création et l'utilisation appropriée d'index peuvent considérablement améliorer les performances des requêtes en accélérant la recherche de données.

Exemple : Créer des indexes sur les colonnes fréquemment utilisées dans les clauses WHERE ou JOIN pour réduire le temps de recherche.

- ## Mise en Cache des Plans d'Exécution :

Définition : SQL Server maintient un cache de plans d'exécution, permettant de réutiliser les plans de requête déjà optimisés.

Exemple : Si une requête identique est soumise à plusieurs reprises, SQL Server peut utiliser le plan d'exécution déjà stocké dans le cache.

- ## Statistiques et Actualisation :

Définition : Les statistiques sur les données aident le moteur de base de données à prendre des décisions éclairées sur la meilleure façon d'exécuter une requête. Les statistiques doivent être régulièrement mises à jour.

Exemple : La commande UPDATE STATISTICS permet de mettre à jour les statistiques, aidant le moteur à prendre des décisions plus précises.

- **Réglage du Serveur :**

Définition : Ajuster les paramètres de configuration du serveur SQL peut avoir un impact significatif sur les performances des requêtes.

Exemple : Ajuster la mémoire allouée au serveur, le degré de parallélisme, ou d'autres paramètres de configuration.

- **Plan d'Exécution en Mode Batch :**

Définition : L'utilisation du mode batch peut améliorer les performances en traitant plusieurs lignes simultanément.

Exemple : Certaines requêtes peuvent être modifiées pour tirer parti du mode batch, améliorant ainsi le parallélisme.

- **Utilisation des Index Columnstore :**

Définition : Pour le traitement massif de données, l'utilisation d'index columnstore peut significativement accélérer les opérations d'analyse.

Exemple : Créer un index columnstore sur des colonnes fréquemment utilisées dans des requêtes analytiques.

- **Ajustement Automatique :**

Définition : SQL Server dispose de mécanismes d'ajustement automatique, tels que le Query Optimizer, qui essaie d'identifier les meilleures stratégies d'exécution.

Exemple : Le Query Optimizer peut choisir d'utiliser un index plutôt qu'une analyse de table complète en fonction de statistiques et de conditions d'exécution.

- **Suivi des Performances :**

Définition : L'utilisation d'outils de suivi des performances permet d'identifier les requêtes lentes et de prendre des mesures correctives.

Exemple : Utiliser des outils comme SQL Server Profiler ou Extended Events pour surveiller et analyser les performances des requêtes.

En résumé, l'optimisation des requêtes dans SQL Server peut se faire manuellement en ajustant les requêtes, en utilisant judicieusement les indexes, en mettant à jour les statistiques, ou automatiquement avec l'aide du Query Optimizer et d'autres mécanismes intégrés. Le suivi continu des performances est essentiel pour identifier les zones nécessitant des améliorations.

Bibliographie

1. Identification du SGBD :

https://fr.wikipedia.org/wiki/Microsoft_SQL_Server

https://learn.microsoft.com/fr-fr/sql/sql-server/?view=sql-server-ver16

2. Architecture fonctionnelle du SGBD :

Architecture SQL Server (expliquée) (guru99.com)

https://www.sqlshack.com/sql-server-2019-new-features-batch-mode-on-rowstore/

3. Le dictionnaire de données :

https://learn.microsoft.com/fr-fr/sql/relational-databases/system-tables/system-base-tables?view=sql-server-ver16

https://learn.microsoft.com/fr-fr/sql/relational-databases/views/views?view=sql-server-ver16

4. Organisation physique :

Groupes de fichiers et fichiers de base de données - SQL Server | Microsoft Learn

https://learn.microsoft.com/fr-fr/sql/relational-databases/pages-and-extents-architecture-guide?view=sql-server-ver16

5. Organisation logique des données :

https://www.quora.com/What-are-MDF-and-NDF-files-in-an-SQL-server

6. Gestion de la concurrence d'accès :

[SQL Server] Transaction : présentation élémentaire et règles ACID « Mohamed A. Cherif - DBMS, BI, Big Data... (wordpress.com)

https://mcherif.wordpress.com/2013/06/06/sql-server-acces-concurrentiels-presentation-des-verrous/

[SQL Server] Accès concurrentiels : présentation des niveaux d'isolation « Mohamed A. Cherif - DBMS, BI, Big Data... (wordpress.com)

7. Gestion des transactions distribuées :

Transactions distribuées - ADO.NET Provider for SQL Server | Microsoft Learn

Configurer les transactions distribuées pour un groupe de disponibilité - SQL Server Always On | Microsoft Learn

Guide de gestion et d'architecture du journal des transactions SQL Server - SQL Server | Microsoft Learn

https://learn.microsoft.com/fr-fr/sql/odbc/reference/develop-app/committing-and-rolling-back-transactions?view=sql-server-ver16

Présentation des transactions - JDBC Driver for SQL Server | Microsoft Learn

Points de contrôle de base de données (SQL Server) - SQL Server | Microsoft Learn

Continuité d'activité et récupération de base de données - SQL Server | Microsoft Learn

8. Gestion de la reprise sur panne

https://learn.microsoft.com/fr-fr/sql/relational-databases/sql-server-transaction-log-architecture-and-management-guide?view=sql-server-ver16

https://learn.microsoft.com/fr-fr/sql/relational-databases/backup-restore/back-up-and-restore-of-sql-server-databases?view=sql-server-ver16

https://learn.microsoft.com/fr-fr/sql/database-engine/configure-windows/buffer-pool-extension?view=sql-server-ver16

https://learn.microsoft.com/fr-fr/sql/relational-databases/maintenance-plans/use-the-maintenance-plan-wizard?view=sql-server-ver16

https://learn.microsoft.com/fr-fr/sql/relational-databases/backup-restore/back-up-a-transaction-log-sql-server?view=sql-server-ver16

https://cloud.google.com/compute/docs/instances/sql-server/disaster-recovery-for-microsoft-sql-server?hl=fr

https://fr.wikipedia.org/wiki/Microsoft_SQL_Server

9. Technique d'indexation

https://learn.microsoft.com/fr-fr/sql/relational-databases/indexes/indexes?view=sql-server-ver16