Projet de recherche sur les SGBD du marché
Dans le cadre du cours FSGBD
(M1 MIAGE UCA NICE)
2020 - 2021

Steven BOUCHE
Armand PREVOT
Pierre GRISERI
Lina BELKARFA
Margaux PARENT
Corentin GARNIER

A rendre au plus tard le 30 Juin 2021

Partie 1 : Evaluation d'un SGBD relationnel du marché

1. Identification du SGBD

Microsoft SQL Server est un **système de gestion de base de données (SGBD) en langage SQL** incorporant entre autres un SGBDR (SGBD relationnel) développé et commercialisé par la société Microsoft.
Développé par Microsoft dont la **première version date du 24 avril 1989** et la **dernière version connue du 4 novembre 2019.**
Développer en **C++, C et C#**, il fonctionne sous les **OS Windows et Linux** (depuis mars 2016), mais il est possible de le lancer sur macOS via Docker, car il en existe une version en téléchargement sur le site de Microsoft.
SQL Server Express est une version d'entrée de gamme gratuite de la base de données, idéale pour l'apprentissage, ainsi que pour la création d'applications de bureau et de petits serveurs jusqu'à 10 Go de données.

A. Modèles de données supportées

Modèles de données principal : Relationnelle

Les systèmes de gestion de bases de données relationnelles (SGBDR) prennent en charge le modèle de données relationnelles (= orienté table). Le schéma d'une table (= schéma de relation) est défini par le nom de la table et un nombre fixe d'attributs avec des types de données fixes. Un enregistrement (=entité) correspond à une ligne de la table et se compose des valeurs de chaque attribut. Une relation consiste donc en un ensemble d'enregistrements uniformes.

Les schémas de table sont générés par normalisation dans le processus de modélisation des données.

Certaines opérations de base sont définies sur les relations :

- opérations ensemblistes classiques (union, intersection et différence)
- Sélection (sélection d'un sous-ensemble d'enregistrements selon certains critères de filtrage des valeurs d'attributs)
- Projection (sélection d'un sous-ensemble d'attributs/colonnes du tableau)
- Join : conjonction spéciale de plusieurs tables comme combinaison du produit cartésien avec sélection et projection.

Ces opérations de base, ainsi que les opérations de création, de modification et de suppression de schémas de table, les opérations de contrôle des transactions et de gestion des utilisateurs sont effectuées au moyen de langages de base de données, SQL étant un standard bien établi pour ces langages.

Modèles de données secondaire : Datastore, Graph, Spatial

Datastore

Les magasins de documents, également appelés systèmes de base de données orientés document, se caractérisent par leur organisation des données sans schéma.

Cela signifie:

- Les enregistrements n'ont pas besoin d'avoir une structure uniforme, c'est-à-dire que des enregistrements différents peuvent avoir des colonnes différentes.
- Les types des valeurs des colonnes individuelles peuvent être différents pour chaque enregistrement.
- Les colonnes peuvent avoir plusieurs valeurs (tableaux).
- Les enregistrements peuvent avoir une structure imbriquée.

Les magasins de documents utilisent souvent des notations internes, qui peuvent être traitées directement dans des applications, principalement JSON. Bien entendu, les documents JSON peuvent également être stockés sous forme de texte pur dans des magasins de valeurs-clés ou des systèmes de bases de données relationnelles . Cela nécessiterait cependant un traitement des structures côté client, ce qui présente l'inconvénient que les fonctionnalités offertes par les magasins de documents (comme les index secondaires) ne sont pas disponibles.

Graph

Les SGBD de graphes, également appelés SGBD orientés graphes ou base de données de graphes, représentent les données dans les structures de graphe sous forme de nœuds et d'arêtes, qui sont des relations entre les nœuds. Ils permettent un traitement facile des données sous cette forme et un calcul simple des propriétés spécifiques du graphe, telles que le nombre d'étapes nécessaires pour passer d'un nœud à un autre.

Les SGBD graphiques ne fournissent généralement pas d'index sur tous les nœuds, l'accès direct aux nœuds en fonction des valeurs d'attribut n'est pas possible dans ces cas.

Spatial

Un SGBD spatial est un système de gestion de base de données capable de stocker, de manipuler et d'interroger efficacement des données spatiales. Les données spatiales représentent des objets dans un espace géométrique, par exemple des points et des polygones.

Les SGBD spatiaux fournissent généralement des types de données dédiés pour stocker des données spatiales et des indices spatiaux pour optimiser l'accès aux ensembles de données spatiales. Les indices spatiaux permettent, par exemple, de récupérer efficacement des points qui se trouvent à une certaine distance d'autres objets. De plus, les SGBD spatiaux fournissent des fonctionnalités pour effectuer des opérations sur des objets ou pour manipuler des objets. Les exemples sont le calcul de distances, la fusion ou l'intersection d'objets et le calcul de propriétés d'objets, telles que les zones de polygones.

Les données géospatiales sont un sous-ensemble important de données spatiales, traitant des données qui décrivent des emplacements sur la surface de la Terre. Les systèmes d'information géographique (SIG) sont capables de travailler avec des données géospatiales.

Les données spatio-temporelles sont une autre variante courante, où les données spatiales sont combinées avec des horodatages, offrant ainsi une autre dimension pour le stockage et la manipulation des données.

B. Versions

Les différences entre SQL Server 2016, 2017 et 2019 :

SQL Server 2016

- **Toujours Encrypté:** la fonctionnalité Always Encrypted protège les données et permet au SQL Server d'effectuer des opérations de données chiffrées afin que les propriétaires puissent protéger leurs données confidentielles à l'aide d'une clé de chiffrement. Cette fonctionnalité garantit que vos données importantes stockées dans la base de données gérée dans le cloud restent chiffrées et protégées.
- **Dissimilation dynamique des données:** cette fonction donne une version illisible de vos données confidentielles à certaines personnes et permet uniquement aux utilisateurs autorisés de les consulter. Il est principalement utilisé pour gérer les informations sur les cartes de crédit et les bases de données similaires.
- **Analyse opérationnelle en temps réel:** l'analyse opérationnelle en temps réel de SQL 2016 prépare votre système à des performances transactionnelles optimales et contribue à augmenter la cohérence de la charge de travail en combinant OLTP en mémoire avec columnstore en mémoire.

SQL Server 2017

- **Conteneurs Windows, Linux et Docker:** vous avez désormais le choix des systèmes d'exploitation que vous utiliserez pour exécuter vos instances de SQL Server 2017.
- **Fonctionnalités de base de données de graphes:** avec la fonctionnalité de base de données de graphes dans SQL Server 2017, vous pouvez désormais stocker et interroger des relations complexes entre les nœuds et les arêtes d'entité plus efficacement.
- **Correction automatique du plan:** la correction automatique du plan garantit des performances continues en détectant et en résolvant les pertes de performances.

SQL Server 2019

- **Clusters Big Data:** cette fonctionnalité vous permet de déployer simultanément plusieurs clusters évolutifs de conteneurs SQL Server, Spark et HDFS s'exécutant sur Kubernetes. Le Big Data Cluster, en tant qu'infrastructure, permet à ces clusters de fonctionner en parallèle, où vous pouvez lire, écrire et traiter des Big Data de Transact-SQL vers Spark. Cela nous permet de combiner et d'analyser facilement des données relationnelles de haute qualité avec de grandes quantités de big data.
- **Prise en charge UTF-8:** SQL Server 2019 prend en charge le système d'encodage de données UTF-8 qui est très populaire. Le codage de caractères UTF-8 est utilisé lors de l'exportation, de l'importation et de la collecte de données au niveau de la base de données et au niveau de la colonne. Il est activé lors de la création ou de la modification du type de tri des objets dans le tri d'objets avec UTF-8. Il est pris en charge pour les types de données char et varchar.

- **Sécurité renforcée:** étant donné que SQL Server est directement concerné par la gestion et l'approvisionnement des bases de données, la sécurité des transactions et des données concernées est l'une des exigences les plus importantes. La sécurité d'accès aux serveurs SQL est gérée par des certificats. La nouvelle fonctionnalité de sécurité SQL Server 2019 inclut la gestion des certificats dans SQL Server Configuration Manager (CTP 2.0). Ce certificat vérifie l'accès sécurisé aux instances SQL Server. La gestion des certificats est désormais dédiée à SQL Server Configuration Manager, simplifiant d'autres tâches.

2. Architecture fonctionnelles du SGBD choisi

A. Architecture

SQL Server est une architecture client-serveur. Le processus MS SQL Server commence par l'envoi d'une requête par l'application cliente. Le serveur SQL accepte, traite et répond à la demande avec des données traitées. Discutons en détail de l'ensemble de l'architecture ci-dessous :

Comme l'illustre le diagramme ci-dessous, l'architecture SQL Server comporte trois composants principaux :

- Couche de protocole
- Moteur relationnel
- Moteur de stockage

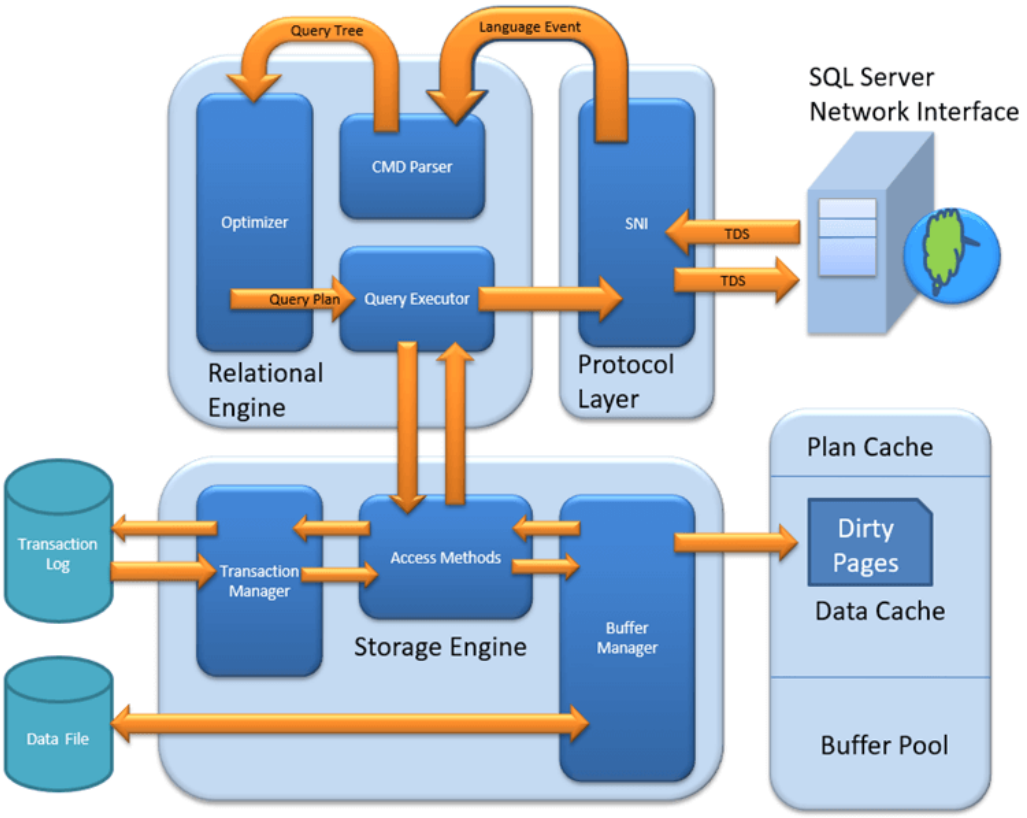
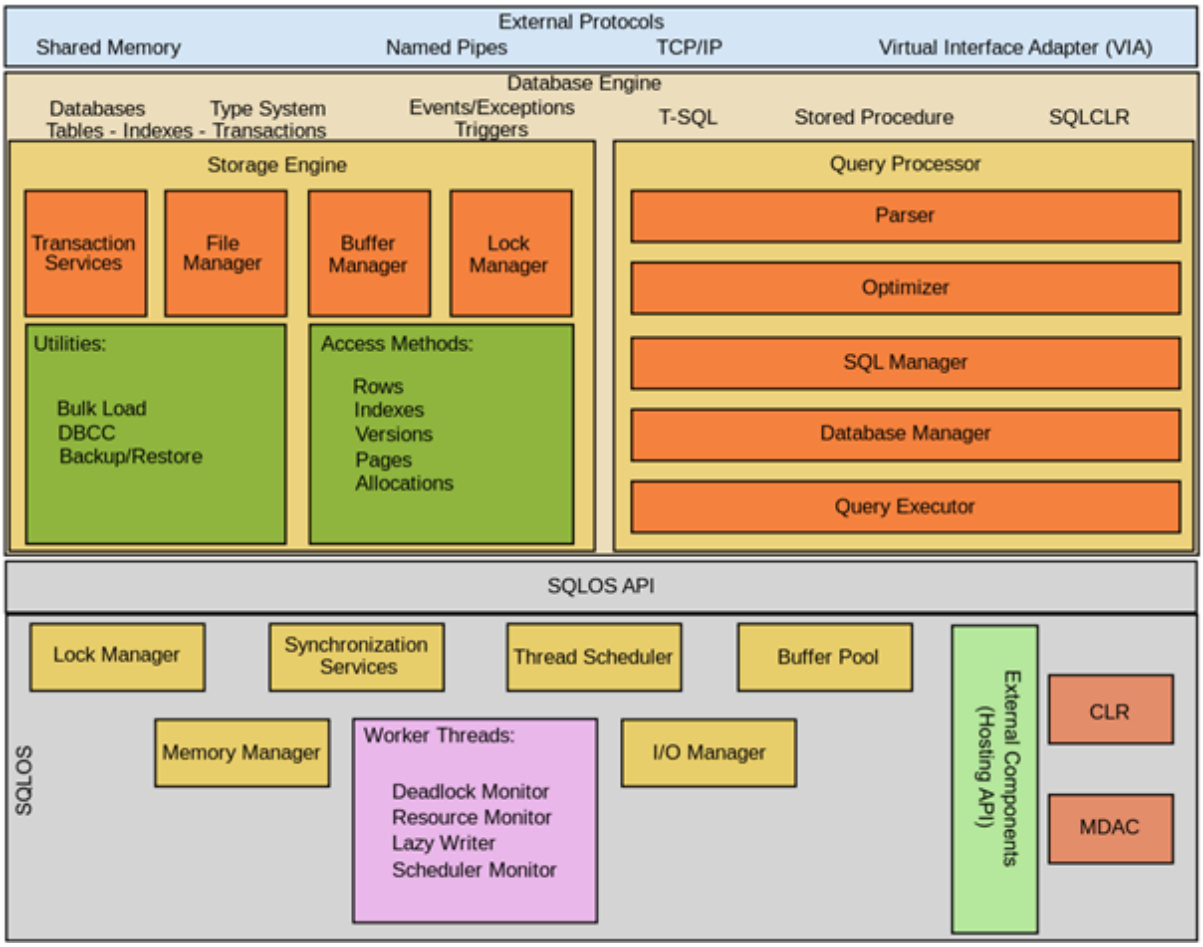


Diagramme d'architecture SQL Server



B. Les caches mémoires et leurs rôles

<u>Architecture mémoire SQL Server</u>

SQL Server acquiert et libère de la mémoire de manière dynamique selon les besoins. En règle générale, un administrateur n'a pas à spécifier la quantité de mémoire à allouer à SQL Server, bien que l'option existe toujours et soit requise dans certains environnements.
L'un des principaux objectifs de conception de tous les logiciels de base de données est de minimiser les E/S disque, car les lectures et écritures sur disque font partie des opérations les plus gourmandes en ressources. SQL Server crée un pool de mémoire tampon pour conserver les pages lues à partir de la base de données. Une grande partie du code de SQL Server est dédiée à la réduction du nombre de lectures et d'écritures physiques entre le disque et le pool de mémoire tampon. SQL Server essaie d'atteindre un équilibre entre deux objectifs :

- Empêchez le pool de mémoire tampon de devenir si volumineux que l'ensemble du système manque de mémoire.
- Minimisez les E/S physiques vers les fichiers de base de données en maximisant la taille du pool de mémoire tampon.

Modification de la gestion mémoire depuis SQL Server 2012 (11.x)

Dans les versions antérieures de SQL Server (SQL Server 2005 (9.x), SQL Server 2008 et SQL Server 2008 R2), l'allocation de mémoire était effectuée à l'aide de cinq mécanismes différents :

- Allocateur de page unique (SPA) , y compris uniquement les allocations de mémoire inférieures ou égales à 8 Ko dans le processus SQL Server. Les options de configuration max server memory (Mo) et min server memory (Mo) déterminent les limites de la mémoire physique consommée par le SPA. Le Buffer Pool était à la fois le mécanisme de SPA et le plus gros consommateur d'allocations d'une seule page.
- Multi-Page Allocator (MPA) , pour les allocations de mémoire qui demandent plus de 8 Ko.
- CLR Allocator , y compris les tas SQL CLR et ses allocations globales qui sont créées lors de l'initialisation du CLR.
- Allocations de mémoire pour les piles de threads dans le processus SQL Server.
- Allocations Windows directes (DWA) , pour les demandes d'allocation de mémoire adressées directement à Windows. Ceux-ci incluent l'utilisation du tas Windows et les allocations virtuelles directes effectuées par les modules qui sont chargés dans le processus SQL Server. Des exemples de telles demandes d'allocation de mémoire incluent les allocations de DLL de procédure stockée étendue, les objets créés à l'aide de procédures Automation (appels sp_OA) et les allocations de fournisseurs de serveurs liés.

À partir de SQL Server 2012 (11.x), les allocations sur une seule page, les allocations sur plusieurs pages et les allocations CLR sont toutes consolidées dans un allocateur de page « Toute taille » , et il est inclus dans les limites de mémoire qui sont contrôlées par la mémoire maximale du serveur (Mo) et les options de configuration de la mémoire minimale du serveur (Mo) . Cette modification a fourni une capacité de dimensionnement plus précise pour toutes les exigences de mémoire qui passent par le gestionnaire de mémoire SQL Server.

À partir de SQL Server 2012 (11.x), SQL Server peut allouer plus de mémoire que la valeur spécifiée dans le paramètre de mémoire maximale du serveur. Ce problème peut se produire lorsque la valeur Total Server Memory (KB) a déjà atteint le paramètre Target Server Memory (KB) (comme spécifié par max server memory). S'il n'y a pas suffisamment de mémoire libre contiguë pour répondre à la demande de demande de mémoire multipages (plus de 8 Ko) en raison de la fragmentation de la mémoire, SQL Server peut effectuer une sur-engagement au lieu de rejeter la demande de mémoire.
Dès que cette allocation est effectuée, la tâche en arrière-plan du moniteur de ressources commencé à signaler à tous les consommateurs de mémoire de libérer la mémoire allouée et essaie d'amener la valeur de la mémoire totale du serveur (Ko) en dessous de la spécification de la mémoire du serveur cible (KB) . Par conséquent, l'utilisation de la mémoire SQL Server peut brièvement dépasser le paramètre de mémoire maximale du serveur. Dans ce cas, la lecture du compteur de performances de la mémoire totale du serveur (Ko) dépassera les paramètres de mémoire maximale du serveur et de mémoire du serveur cible (Ko) .
Ce comportement est généralement observé lors des opérations suivantes :

- Grandes requêtes d'index Columnstore.
- Les (re)constructions d'index Columnstore, qui utilisent de gros volumes de mémoire pour effectuer des opérations de hachage et de tri.
- Opérations de sauvegarde qui nécessitent de grandes mémoires tampons.
- Opérations de traçage qui doivent stocker des paramètres d'entrée volumineux.

Gestion dynamique de la mémoire

Le comportement de gestion de la mémoire par défaut du moteur de base de données SQL Server consiste à acquérir autant de mémoire que nécessaire sans créer de pénurie de mémoire sur le système. Pour ce faire, le moteur de base de données SQL Server utilise les API de notification de mémoire dans Microsoft Windows.

Lorsque SQL Server utilise la mémoire de manière dynamique, il interroge périodiquement le système pour déterminer la quantité de mémoire disponible. Le maintien de cette mémoire libre empêche le système d'exploitation (OS) de paginer. Si moins de mémoire est disponible, SQL Server libère de la mémoire pour le système d'exploitation. Si plus de mémoire est disponible, SQL Server peut allouer plus de mémoire. SQL Server ajoute de la mémoire uniquement lorsque sa charge de travail nécessite plus de mémoire ; un serveur au repos n'augmente pas la taille de son espace d'adressage virtuel.

La mémoire maximale du serveur contrôle l'allocation de mémoire SQL Server, la mémoire de compilation, tous les caches (y compris le pool de mémoire tampon), les allocations de mémoire d'exécution de requête , la mémoire du gestionnaire de verrouillage et la mémoire CLR 1 (essentiellement tout commis de mémoire trouvé dans sys.dm_os_memory_clerks).

SQL Server utilise l'API de notification de mémoire QueryMemoryResourceNotification pour déterminer quand le gestionnaire de mémoire SQL Server peut allouer de la mémoire et libérer de la mémoire.

Lorsque SQL Server démarre, il calcule la taille de l'espace d'adressage virtuel pour le pool de mémoire tampon en fonction d'un certain nombre de paramètres tels que la quantité de mémoire physique sur le système, le nombre de threads de serveur et divers paramètres de démarrage. SQL Server réserve la quantité calculée de son espace d'adressage virtuel de processus pour le pool de mémoire tampon, mais il acquiert (valide) uniquement la quantité de mémoire physique requise pour la charge actuelle.

L'instance continue ensuite à acquérir de la mémoire selon les besoins pour prendre en charge la charge de travail. Au fur et à mesure que de plus en plus d'utilisateurs se connectent et exécutent des requêtes, SQL Server acquiert la mémoire physique supplémentaire à la demande. Une instance SQL Server continue d'acquérir de la mémoire physique jusqu'à ce qu'elle atteigne sa cible d'allocation de mémoire maximale du serveur ou que le système d'exploitation indique qu'il n'y a plus d'excès de mémoire libre ; il libère de la mémoire lorsqu'il a plus que le paramètre de mémoire minimale du serveur, et le système d'exploitation indique qu'il y a un manque de mémoire libre.

Comme d'autres applications sont démarrées sur un ordinateur exécutant une instance de SQL Server, elles consomment de la mémoire et la quantité de mémoire physique libre tombe en dessous de la cible SQL Server. L'instance de SQL Server ajuste sa consommation mémoire. Si une autre application est arrêtée et que davantage de mémoire devient disponible, l'instance SQL Server augmente la taille de son allocation de mémoire. SQL Server peut libérer et acquérir plusieurs mégaoctets de mémoire chaque seconde, ce qui lui permet de s'adapter rapidement aux changements d'allocation de mémoire.

Gestion tampons

L'objectif principal d'une base de données SQL Server est de stocker et de récupérer des données. Les E/S de disque intensives sont donc une caractéristique essentielle du moteur de base de données. Et comme les opérations d'E/S sur disque peuvent consommer de nombreuses ressources et prendre un temps relativement long, SQL Server se concentre sur l'efficacité des E/S. La gestion des tampons est un élément clé pour atteindre cette efficacité. Le composant de gestion des tampons se compose de deux mécanismes : le gestionnaire de tampons pour accéder et mettre à jour les pages de la base de données, et le cache de tampons (également appelé pool de tampons), pour réduire les E/S des fichiers de la base de données.

Un tampon est une page de 8 Ko en mémoire, de la même taille qu'une page de données ou d'index. Ainsi, le cache tampon est divisé en pages de 8 Ko. Le gestionnaire de tampon gère les fonctions de lecture de données ou de pages d'index à partir des fichiers disque de la base de données dans le cache tampon et de réécriture des pages modifiées sur le disque. Une page reste dans le cache tampon jusqu'à ce que le gestionnaire de tampon ait besoin de la zone tampon pour lire plus de données. Les données ne sont réécrites sur le disque que si elles sont modifiées. Les données du cache tampon peuvent être modifiées plusieurs fois avant d'être réécrites sur le disque. Pour plus d'informations, voir Lecture de pages et Écriture de pages .

Lorsque SQL Server démarre, il calcule la taille de l'espace d'adressage virtuel pour le cache tampon en fonction d'un certain nombre de paramètres tels que la quantité de mémoire physique sur le système, le nombre configuré de threads de serveur maximum et divers paramètres de démarrage. SQL Server réserve cette quantité calculée de son espace d'adressage virtuel de processus (appelé la cible de mémoire) pour le cache de tampon, mais il acquiert (valide) uniquement la quantité requise de mémoire physique pour la charge actuelle. Vous pouvez interroger les colonnes bpool_commit_target et bpool_committed dans la vue catalogue sys.dm_os_sys_info pour renvoyer respectivement le nombre de pages réservées comme cible de mémoire et le nombre de pages actuellement validées dans le cache de tampon.

L'intervalle entre le démarrage de SQL Server et le moment où le cache tampon obtient sa cible de mémoire est appelé accélération. Pendant ce temps, les demandes de lecture remplissent les tampons selon les besoins. Par exemple, une seule demande de lecture de page de 8 Ko remplit une seule page de tampon. Cela signifie que la montée en puissance dépend du nombre et du type de demandes des clients. La montée en puissance est accélérée en transformant les demandes de lecture d'une seule page en demandes alignées de huit pages (constituant une extension). Cela permet à la montée en puissance de se terminer beaucoup plus rapidement, en particulier sur les machines disposant de beaucoup de mémoire.

Étant donné que le gestionnaire de mémoire tampon utilisé la majeure partie de la mémoire dans le processus SQL Server, il coopère avec le gestionnaire de mémoire pour permettre à d'autres composants d'utiliser ses mémoires tampon. Le gestionnaire de tampons interagit principalement avec les composants suivants :

- Gestionnaire de ressources pour contrôler l'utilisation globale de la mémoire et, dans les plates-formes 32 bits, pour contrôler l'utilisation de l'espace d'adressage.
- Gestionnaire de base de données et système d'exploitation SQL Server (SQLOS) pour les opérations d'E/S de fichiers de bas niveau.
- Gestionnaire de journaux pour la journalisation en écriture anticipée.

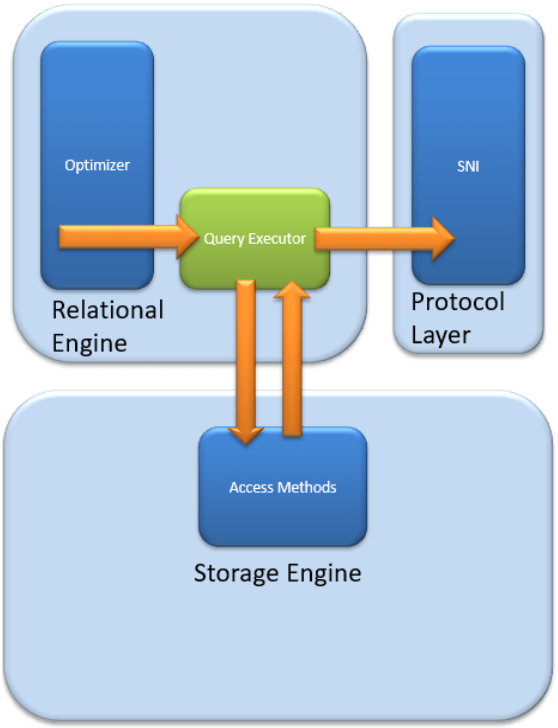
C. Gestion des connexions

Lorsqu'une application communique avec le moteur de base de données, les interfaces de programmation d'application (API) exposées par la couche de protocole formate la communication à l'aide d'un format défini par Microsoft appelé *paquet de flux de données tabulaires (TDS)* . La couche de protocole SQL Server Network Interface (SNI) sur les ordinateurs serveur et client encapsule le paquet TDS dans un protocole de communication standard, tel que TCP / IP ou Named Pipes. Du côté serveur de la communication, les bibliothèques réseau font partie du moteur de base de données. Côté client, les bibliothèques réseau font partie de SQL Native Client. La configuration du client et l'instance de SQL Server déterminent le protocole utilisé.

Vous pouvez configurer SQL Server pour prendre en charge plusieurs protocoles simultanément, provenant de différents clients. Chaque client se connecte à SQL Server avec un seul protocole. Si le programme client ne sait pas sur quels protocoles SQL Server écoute, vous pouvez configurer le client pour tenter plusieurs protocoles de manière séquentielle. Les protocoles suivants sont disponibles:

- **Mémoire partagée** . Le protocole le plus simple à utiliser, sans paramètres configurables. Les clients utilisant le protocole de mémoire partagée peuvent se connecter uniquement à une instance SQL Server exécutée sur le même ordinateur, ce protocole n'est donc pas utile pour la plupart des activités de base de données. Utilisez ce protocole pour le dépannage lorsque vous pensez que les autres protocoles sont configurés de manière incorrecte. Les clients utilisant MDAC 2.8 ou version antérieure ne peuvent pas utiliser le protocole de mémoire partagée. Si une telle connexion est tentée, le client passe au protocole Named Pipes.
- **Nommé Pipes** . Un protocole développé pour les réseaux locaux (LAN). Une partie de la mémoire est utilisée par un processus pour transmettre des informations à un autre processus, de sorte que la sortie de l'un est l'entrée de l'autre. Le deuxième processus peut être local (sur le même ordinateur que le premier) ou distant (sur un ordinateur du réseau).
- **TCP / IP** . Le protocole le plus utilisé sur Internet. TCP / IP peut communiquer sur des réseaux informatiques interconnectés avec diverses architectures matérielles et systèmes d'exploitation. Il comprend des normes de routage du trafic réseau et offre des fonctionnalités de sécurité avancées. L'activation de SQL Server pour utiliser TCP / IP nécessite le plus d'efforts de configuration, mais la plupart des ordinateurs en réseau sont déjà correctement configurés.

D. Traitement et exécution des requêtes



L'exécuteur de requêtes appelle la méthode d'accès. Il fournit un plan d'exécution pour la logique d'extraction de données requise pour l'exécution. Une fois les données reçues du moteur de stockage, le résultat est publié dans la couche de protocole. Enfin, les données sont envoyées à l'utilisateur final.

Le Moteur de base de données SQL Server traite les requêtes sur diverses architectures de stockage des données, telles que des tables locales, des tables partitionnées et des tables distribuées sur plusieurs serveurs. Les rubriques suivantes expliquent comment SQL Server traite les requêtes et optimise leur réutilisation grâce à la mise en cache du plan d'exécution.

Le Moteur de base de données SQL Server peut traiter les instructions Transact-SQL selon deux modes de traitement distincts :

- Exécution en mode ligne
- Exécution en mode batch

Exécution en mode ligne

L'exécution en mode ligne est une méthode de traitement des requêtes utilisée avec les tables traditionnelles des SGBDR, où les données sont stockées à un format ligne. Quand une requête est exécutée et accède aux données de tables contenant des lignes, les opérateurs de l'arborescence d'exécution et les opérateurs enfants lisent chaque ligne nécessaire, dans toutes les colonnes qui sont spécifiées dans le schéma de table. Pour chaque ligne lue, SQL Server récupère ensuite les colonnes qui sont nécessaires pour le jeu de résultats, telles qu'elles sont référencées par une instruction SELECT, un prédicat JOIN ou un prédicat de filtre.

Exécution en mode batch

L'exécution en mode batch est une méthode de traitement des requêtes utilisée pour traiter plusieurs lignes ensemble (d'où le terme « batch »). Chaque colonne d'un batch est stockée sous forme de vecteur dans une zone distincte de la mémoire : ainsi, le traitement en mode batch est basé sur les vecteurs. Le traitement en mode batch utilise aussi des algorithmes qui sont optimisés pour les processeurs multicœurs et un débit mémoire amélioré qui se trouve sur le matériel moderne.

L'exécution en mode batch est étroitement intégrée au format de stockage columnstore et optimisée pour celui-ci. Le traitement en mode batch s'effectue quand c'est possible sur des données compressées et il élimine les opérateurs d'échange utilisés par le traitement en mode ligne. Le résultat est un meilleur parallélisme et des performances plus rapides.
Quand une requête est exécutée en mode batch et qu'elle accède à des données dans des index columnstore, les opérateurs de l'arborescence d'exécution et les opérateurs enfants lisent plusieurs lignes à la fois dans les segments de colonne. SQL Server lit seulement les colonnes nécessaires pour le résultat, telles qu'elles sont référencées par une instruction SELECT, un prédicat JOIN ou un prédicat de filtre.

Traitement instruction Select

Les étapes permettant à SQL Server de traiter une instruction SELECT unique sont les suivantes :

- L'analyseur examine l'instruction SELECT et la décompose en unités logiques telles que mots clé, expressions, opérateurs et identificateurs.

- Un arbre de requêtes, également appelé arbre de séquence, est créé pour décrire les étapes logiques nécessaires à la transformation des données source au format requis par le jeu de résultats.

- L'optimiseur de requête analyse plusieurs méthodes d'accès aux tables source. Il choisit ensuite la série d'étapes qui retournent les résultats le plus rapidement tout en consommant moins de ressources. L'arbre de requêtes est mis à jour pour enregistrer cette série exacte d'étapes. La version optimisée finale de l'arbre de requêtes est nommée plan d'exécution.

- Le moteur relationnel lance le plan d'exécution. Pendant le traitement des étapes qui requièrent des données issues des tables de base, le moteur relationnel demande que le moteur de stockage transmette les données des ensembles de lignes demandés à partir du moteur relationnel.

- Le moteur relationnel traite les données retournées du moteur de stockage dans le format défini pour le jeu de résultats et retourne ce jeu au client.

E. Architecture des threads et des tâches

Dans l'étendue de SQL Server, une requête est la représentation logique d'une requête ou d'un lot. Une requête représente également des opérations requises par des threads de système, telles qu'un point de contrôle ou un enregistreur de journal. Les requêtes existent dans différents états pendant toute leur durée de vie et peuvent accumuler les attentes lorsque les ressources requises pour exécuter la requête, par exemple des verrouiller ou verrous, ne sont pas disponibles.

Une tâche représente l'unité de travail à effectuer pour exécuter la requête. Une ou plusieurs tâches peuvent être attribuées à une seule requête.

- Les requêtes parallèles comportent plusieurs tâches actives qui sont exécutées simultanément plutôt qu'en série, avec une tâche parente (ou une tâche qui coordonne) et plusieurs tâches enfants. Un plan d'exécution pour une requête parallèle peut avoir des branches en série - des zones du plan avec des opérateurs qui n'exécutent pas en parallèle. La tâche parente est également chargée de l'exécution de ces opérateurs en série.
- Les requêtes en série n'auront qu'une seule tâche active à un moment donné de leur exécution.
  Les tâches existent dans différents états pendant toute leur durée de vie.

Un thread de travail SQL Server, également appelé Worker ou thread, est une représentation logique d'un thread de système d'exploitation. Lors de l'exécution de requêtes en série, la Moteur de base de données SQL Server génère un Worker pour exécuter la tâche active (1:1). Lors de l'exécution de requêtes parallèles en mode en ligne, le Moteur de base de données SQL Server affecte un worker pour coordonner les travailleurs enfants chargés de l'exécution des tâches qui leur sont attribuées (1:1), appelées le thread parent (ou coordonner le thread). Une tâche parente est associée à un thread parent. Le thread parent est le point d'entrée de la demande et existe même avant que le moteur n'analyse une requête. Les principales responsabilités du thread parent sont les suivantes :

- Coordonner une analyse parallèle.
- Démarrer les threads de travail parallèles enfants.
- Collecter les lignes des threads parallèles et les envoyer au client.
- Effectuer des agrégations locales et globales.

Le nombre de threads de travail générés pour chaque tâche dépend de ce qui suit :

- Si la requête était éligible pour le parallélisme comme déterminé par l'optimiseur de requête.
- Quel est le degré de parallélisme (DOP) effectif disponible dans le système en fonction de la charge actuelle. Cela peut diverger du degré de parallélisme estimé, qui est basé sur la configuration du serveur pour le degré maximal de parallélisme (MAXDOP). Par exemple, la configuration du serveur pour MAXDOP peut être 8, mais le DOP disponible au moment de l'exécution peut être de 2 seulement, ce qui affecte les performances des requêtes.

Un planificateur, également appelé planificateur SOS, gère les threads de travail nécessitant un temps de traitement pour effectuer le travail résultant de tâches. Chaque planificateur est mappé à un processeur (UC) individuel. La durée pendant laquelle un Worker peut rester actif dans un planificateur est appelée quantum de système d'exploitation, avec un maximum de 4 ms. Une fois que son temps de quantum a expiré, un thread de travail consacre son temps à d'autres threads

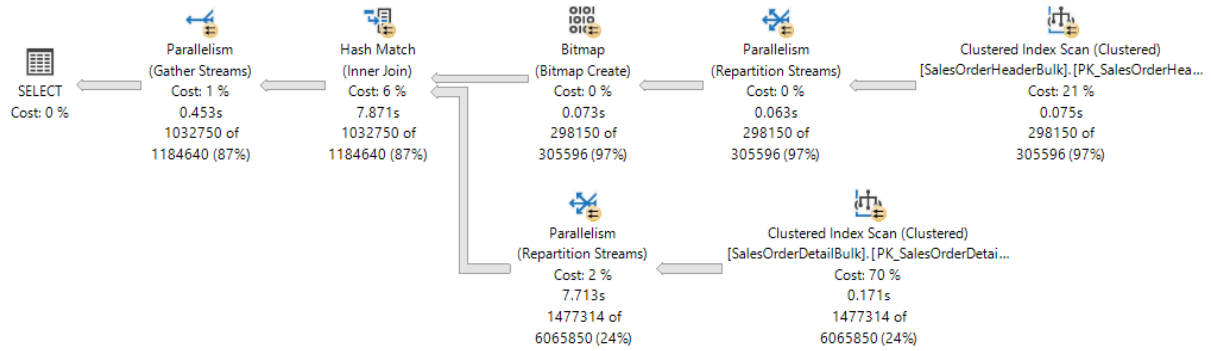
ayant besoin d'accéder aux ressources de l'UC et change d'état. Cette coopération entre les Workers pour optimiser l'accès aux ressources de l'UC est appelée planification coopérative, également connue sous le nom de planification non préemptive. À son tour, la modification de l'état du Worker est propagée à la tâche associée à ce Worker ainsi qu'à la requête associée à la tâche.

En résumé, une requête peut générer une ou plusieurs tâches pour effectuer des unités de travail. Chaque tâche est assignée à un thread de travail qui est responsable de l'exécution de la tâche. Chaque thread de travail doit être planifié (placé sur un planificateur) pour l'exécution active de la tâche.

Planification de tâches parallèles

Imaginez une SQL Server configurée avec MaxDOP 8, et l'affinité du processeur est configurée pour 24 UC (planificateurs) sur les nœuds NUMA 0 et 1. Les planificateurs de 0 à 11 appartiennent au nœud NUMA 0, les planificateurs de 12 à 23 appartiennent au nœud NUMA 1. Une application envoie la requête suivante (request) au Moteur de base de données :

Le plan d'exécution affiche une jointure de hachage entre deux tables, et chacun des opérateurs exécutés en parallèle, comme indiqué par le cercle jaune avec deux flèches. Chaque opérateur de parallélisme est une branche différente dans le plan. Par conséquent, il existe trois branches dans le plan d'exécution ci-dessous.

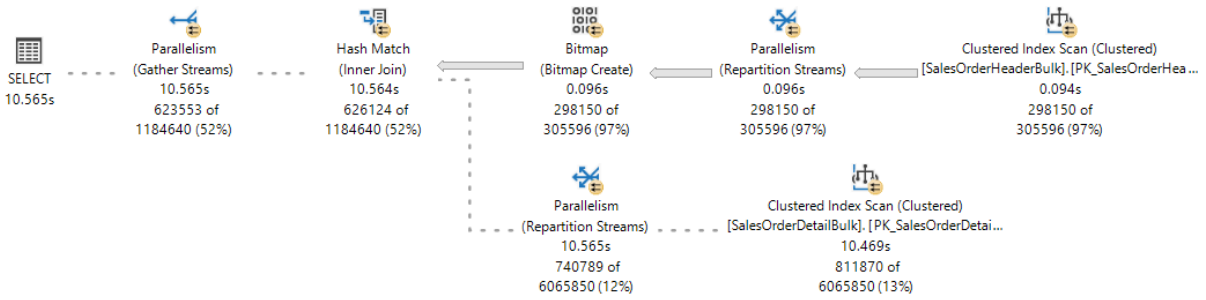


Bien qu'il y ait trois branches dans le plan d'exécution, à tout moment pendant l'exécution, seules deux branches peuvent s'exécuter simultanément dans ce plan d'exécution :

1. La branche dans laquelle une analyse d'index cluster est utilisée sur Sales.SalesOrderHeaderBulk (entrée de build de la jointure) s'exécute seule.
2. Ensuite, la branche dans laquelle une Analyse d'index cluster est utilisée sur le Sales.SalesOrderDetailBulk (entrée build de la jointure) s'exécute simultanément avec la branche dans laquelle le Bitmap a été créé et où le Hash Match s'exécute.

La réservation de thread garantit que le Moteur de base de données dispose de suffisamment de threads de travail pour effectuer toutes les tâches qui seront nécessaires pour la requête. Les threads peuvent être réservés sur plusieurs nœuds NUMA ou être réservés dans un seul nœud NUMA. La réservation de thread est effectuée au moment de l'exécution avant le démarrage de l'exécution, et dépend de la charge du planificateur. Le nombre de threads de travail réservés est dérivé de façon générique à partir de la formule d'exécution DOP * parallélisme de Runtime et exclut le thread de travail parent. Chaque branche est limitée à un nombre de threads de travail égaux à MaxDOP. Dans

cet exemple, il existe deux branches simultanées et MaxDOP a pour valeur 8, par conséquent 2 * 8 = 16.
Pour référence, observez le plan d'exécution en direct à partir de Statistiques des Requêtes en direct, où une branche est terminée et que deux branches s'exécutent simultanément.



Un thread de travail ne peut rester actif dans le planificateur que pour la durée de son quantum (4 ms) et doit générer son planificateur après que ce quantum se soit écoulé, de sorte qu'un thread de travail affecté à une autre tâche peut devenir actif. Quand le quantum d'un worker arrive à expiration et qu'il n'est plus actif, la tâche correspondante est placée dans une file d'attente FIFO dans un état EXÉCUTABLE, en supposant que la tâche ne nécessitera pas l'accès à des ressources qui ne sont pas disponibles pour le moment, comme un verrou ou une serrure, auquel cas la tâche serait placée dans un état SUSPENDU au lieu d'EXÉCUTABLE, jusqu'à ce que ces ressources soient disponibles.

En résumé, une demande parallèle génère plusieurs tâches. Chaque tâche doit être affectée à un seul thread de travail. Chaque thread de travail doit être affecté à un seul planificateur. Ainsi, le nombre de planificateurs en cours d'utilisation ne peut pas dépasser le nombre de tâches parallèles par branche, que définit l'indicateur de requête ou la configuration MaxDOP. Le thread de coordination ne contribue pas à la limite MaxDOP.

F. Architecture du journal des transactions

Chaque base de données SQL Server possède un journal des transactions qui enregistre toutes les transactions et les modifications apportées par chacune d'entre elles. Le journal des transactions est un composant essentiel de la base de données et, en cas de défaillance du système, vous pouvez en avoir besoin pour rétablir la cohérence de la base de données. Ce guide contient des informations sur l'architecture physique et logique du journal des transactions. Ces informations pourront vous aider à gérer plus efficacement les journaux des transactions.

Architecture logique du journal des transactions

Le journal des transactions de SQL Server fonctionne de façon logique comme s'il s'agissait d'une chaîne d'enregistrements de journal. Chacun de ces enregistrements est identifié par un numéro séquentiel dans le journal (LSN). Chaque nouvel enregistrement est écrit à la fin logique du journal avec un LSN supérieur à celui de l'enregistrement qui le précède. Les enregistrements de journal sont stockés dans une séquence en série à mesure qu'ils sont créés de sorte que si LSN2 est supérieur à

LSN1, la modification décrite par l'enregistrement de journal référencé par LSN2 se produit après la modification décrite par le numéro LSN1 d'enregistrement de journal. Chacun d'eux contient l'ID de la transaction à laquelle il appartient. Pour chaque transaction, tous les enregistrements de journal associés sont reliés de façon individuelle dans une chaîne grâce aux pointeurs arrière qui accélèrent la restauration de la transaction.

Les enregistrements de journal relatifs aux modifications de données consignent soit l'opération logique effectuée, soit les images avant/après des données modifiées. L'image avant est une copie des données avant que l'opération n'ait été effectuée, tandis que l'image après est une copie des données après que l'opération a été effectuée.

Les étapes pour récupérer une opération dépendent du type de journal d'enregistrement :

- Opération logique enregistrée
    - Si vous repositionnez l'opération logique avant, elle est effectuée à nouveau.
    - Si vous annulez l'opération logique, l'opération inverse est effectuée.
- Image avant et après enregistrées
    - Si vous repositionnez l'opération avant, l'image après est appliquée.
    - Si vous annulez l'opération, l'image avant est appliquée.

De nombreux types d'opérations sont enregistrés dans le journal des transactions. Ces opérations comprennent :

- Le début et la fin de chaque transaction.
- Chaque modification de données (insertion, mise à jour ou suppression). Il s'agit des modifications apportées aux tables, y compris les tables système, par les procédures stockées système ou les instructions DDL (Data Definition Language).
- Chaque allocation et désallocation de page et d'étendue.
- Création ou suppression d'une table ou d'un index.

Les opérations de restauration sont également consignées dans le journal. Chaque transaction réserve de l'espace dans le journal des transactions afin qu'il existe suffisamment d'espace journal pour prendre en charge une restauration déclenchée par une instruction de restauration explicite ou par la détection d'une erreur. Le volume d'espace réservé dépend des opérations effectuées dans la transaction, mais il est généralement égal au volume d'espace utilisé pour la journalisation de chaque opération. Cet espace réservé est libéré lorsque la transaction est terminée.
La section du fichier journal comprise entre le premier enregistrement de journal nécessaire à une restauration portant sur l'ensemble de la base de données et la fin du journal représente la partie active du journal, le journal actif ou la fin du journal. Cette section est indispensable pour procéder à une récupération complète de la base de données. Aucune partie de ce journal actif ne peut être tronquée. Le LSN (numéro séquentiel dans le journal) de ce premier enregistrement est le LSN de récupération minimum (MinLSN) .
Les sauvegardes différentielles et de journaux font passer la base de données restaurée à une date ultérieure qui correspond à un numéro LSN supérieur.

Architecture physique du journal des transactions

Le journal des transactions d'une base de données s'étend sur un ou plusieurs fichiers physiques. D'un point de vue conceptuel, le fichier journal est une chaîne d'enregistrements. D'un point de vue physique, la séquence des enregistrements du journal est stockée de façon efficace dans l'ensemble

de fichiers physiques qui implémente le journal des transactions. Chaque base de données doit posséder au moins un fichier journal.

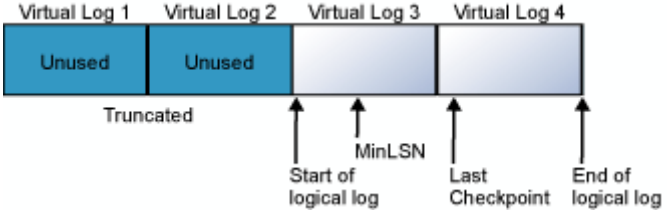
Le Moteur de base de données SQL Server divise chaque fichier journal physique en un certain nombre de fichiers journaux virtuels. La taille et le nombre de ces fichiers journaux virtuels sont variables. Le Moteur de base de données choisit dynamiquement la taille des fichiers journaux virtuels en créant ou en étendant des fichiers journaux. Le Moteur de base de données essaie de ne conserver qu'un petit nombre de fichiers virtuels. Après une extension du fichier journal, la taille des fichiers virtuels est la somme de la taille du journal existant et de la taille du nouvel incrément de fichier. La taille et le nombre des fichiers journaux virtuels ne peuvent être ni configurés, ni définis par les administrateurs.

Architecture physique du journal des transactions

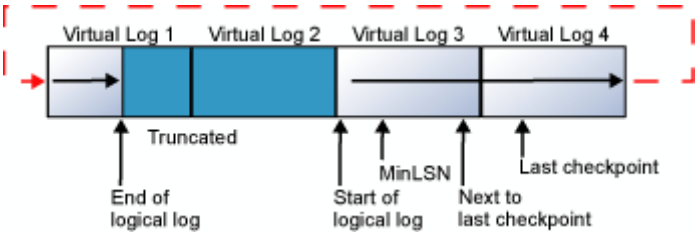
Le journal des transactions d'une base de données s'étend sur un ou plusieurs fichiers physiques. D'un point de vue conceptuel, le fichier journal est une chaîne d'enregistrements. D'un point de vue physique, la séquence des enregistrements du journal est stockée de façon efficace dans l'ensemble de fichiers physiques qui implémente le journal des transactions. Chaque base de données doit posséder au moins un fichier journal.

Le Moteur de base de données SQL Server divise chaque fichier journal physique en un certain nombre de fichiers journaux virtuels. La taille et le nombre de ces fichiers journaux virtuels sont variables. Le Moteur de base de données choisit dynamiquement la taille des fichiers journaux virtuels en créant ou en étendant des fichiers journaux. Le Moteur de base de données essaie de ne conserver qu'un petit nombre de fichiers virtuels. Après une extension du fichier journal, la taille des fichiers virtuels est la somme de la taille du journal existant et de la taille du nouvel incrément de fichier. La taille et le nombre des fichiers journaux virtuels ne peuvent être ni configurés, ni définis par les administrateurs.

Le journal des transactions est un fichier cumulatif. Considérons, par exemple, une base de données possédant un fichier journal physique divisé en quatre fichiers journaux virtuels. Lors de la création de la base de données, le fichier journal logique commence au début du fichier journal physique. Les nouveaux enregistrements du journal sont ajoutés à la fin du journal logique, qui s'étend vers la fin du journal physique. Le fait de tronquer le journal permet de libérer tous les journaux virtuels dont les enregistrements précèdent tous le MinLSN (numéro séquentiel dans le journal minimum). Le MinLSN est le numéro séquentiel dans le journal du plus ancien enregistrement du journal requis pour une opération de restauration réussie de l'ensemble de la base de données. Le journal des transactions de la base de données exemple ressemblerait à celui de l'illustration suivante :



Lorsque la fin du journal logique atteint la fin du fichier journal physique, le nouvel enregistrement du journal revient au début du fichier journal physique.



Le cycle se répète indéfiniment tant que la fin du journal logique n'a pas atteint le début du journal logique. Si les anciens enregistrements du journal sont tronqués suffisamment souvent pour laisser de la place à tous les nouveaux enregistrements créés jusqu'au point de contrôle suivant, le journal ne se remplit jamais. Si la fin du journal logique atteint le début du journal logique, l'une ou l'autre des situations suivantes se produit :

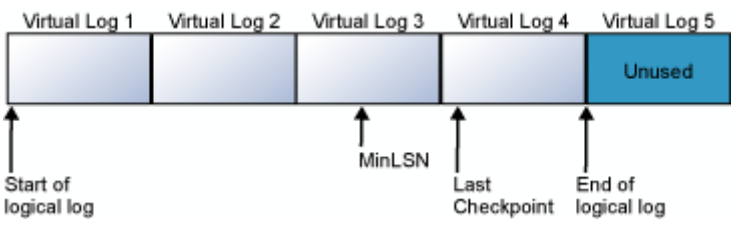
- Si le paramètre FILEGROWTH est activé pour le journal et que l'espace disque est suffisant, le fichier s'étend en fonction de la taille spécifiée dans le paramètre growth_increment, les nouveaux enregistrements du journal étant ajoutés à l'extension.
- Si le paramètre FILEGROWTH n'est pas activé ou si l'espace disque réservé au fichier journal est inférieur à la taille spécifiée dans le paramètre growth_increment, l'erreur 9002 est générée.

Si le journal contient plusieurs fichiers journaux physiques, le journal logique va se déplacer dans tous les fichiers journaux physiques avant de revenir au début du premier fichier journal physique.

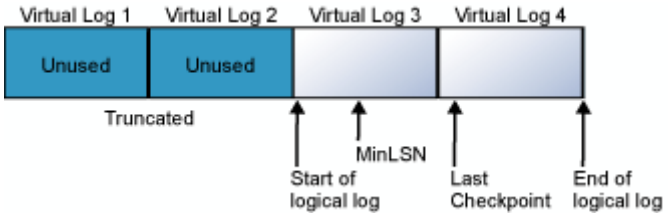
Troncation de journal

La troncation du journal est essentielle pour empêcher que le journal se remplisse. La troncation du journal supprime les fichiers journaux virtuels inactifs du journal des transactions logiques d'une base de données SQL Server , ce qui libère de l'espace dans le journal logique de façon à ce qu'il soit réutilisé par le journal des transactions physique. Si un journal des transactions n'était jamais tronqué, il finirait par occuper tout l'espace disque alloué à ses fichiers journaux physiques. Toutefois, une opération de contrôle est requise avant que le journal des transactions puisse être tronqué. Un point de contrôle écrit les pages modifiées en mémoire actuelles (appelées pages de modifications) et les informations du journal des transactions de la mémoire vers le disque. Lorsque le point de contrôle est créé, la partie inactive du journal des transactions est marquée comme réutilisable. Après cela, elle peut être libérée par troncation du journal.

Les illustrations suivantes montrent un journal des transactions avant et après une troncation. La première illustration montre un journal des transactions qui n'a jamais été tronqué. Actuellement, quatre fichiers journaux virtuels sont utilisés par le journal logique. Le journal logique commence avant le premier fichier journal virtuel et se termine au journal virtuel 4. L'enregistrement NSEmin se trouve dans le journal virtuel 3. Les journaux virtuels 1 et 2 contiennent uniquement des enregistrements de journal inactifs. Ces enregistrements peuvent être tronqués. Le journal virtuel 5 est encore inutilisé et ne fait pas partie du journal logique actuel.



La deuxième illustration montre le journal après sa troncation. Les journaux virtuels 1 et 2 ont été libérés en vue de leur réutilisation. Le journal logique commence désormais au début du journal virtuel 3. Le journal virtuel 5 est encore inutilisé et ne fait pas partie du journal logique actuel.



La troncation du journal se produit automatiquement après les événements suivants, à moins qu'elle ne soit retardée pour une raison quelconque :

- En mode de récupération simple, après un point de contrôle.
- En mode de restauration complète ou en mode de récupération utilisant les journaux de transactions, après une sauvegarde du journal, si un point de contrôle s'est produit depuis la dernière sauvegarde.

Journal des transactions à écriture anticipée

Cette section décrit le rôle que joue le journal des transactions à écriture anticipée (journal WAL) au niveau de l'enregistrement sur disque des modifications apportées aux données. SQL Server utilise un algorithme WAL (write-ahead logging) qui garantit qu'aucune modification de données n'est écrite sur le disque avant l'écriture du journal associé sur celui-ci. Ainsi, les propriétés ACID (Atomicité, Cohérence, Isolation et Durabilité) d'une transaction sont conservées.
Pour comprendre le fonctionnement du journal à écriture anticipée, il est important que vous sachiez comment les données modifiées sont écrites sur le disque. SQL Server gère un cache des tampons dans lequel il lit les pages de données lorsque celles-ci doivent être extraites. Lorsqu'une page est modifiée dans le cache des tampons, elle n'est pas réécrite immédiatement sur le disque, mais elle est marquée comme erronée. Une page peut avoir plusieurs écritures logiques avant son écriture physique sur le disque. Pour chaque écriture logique, un enregistrement du journal des transactions est inséré dans le cache du journal qui enregistre la modification. L'enregistrement doit être écrit sur le disque avant que la page de modifications associée n'ait été supprimée du cache et écrite sur le disque. Le processus de point de contrôle analyse régulièrement le cache à la recherche de tampons contenant des pages issues d'une base de données spécifiée et écrit toutes les pages de modifications sur le disque. Les points de contrôle permettent une récupération ultérieure du système en créant un point où toutes les pages de modifications sont effectivement écrites sur le disque.
Le processus d'écriture d'une page de données modifiée, du cache des tampons vers le disque, porte le nom de vidage. SQL Server possède une logique qui empêche la suppression d'une page de modifications avant que l'enregistrement du journal associé n'ait été écrit. Les enregistrements de journal sont écrits sur le disque quand les tampons de journaux sont vidés. Cela se produit chaque fois qu'une transaction est validée ou que les tampons de journaux sont saturés.

Sauvegardes du journal de transactions

Cette section présente les concepts sur la sauvegarde et la restauration (application) des journaux de transactions. En mode de récupération complète et en mode de récupération utilisant les journaux

de transactions, la sauvegarde régulière des journaux de transactions (sauvegardes des journaux) est indispensable pour pouvoir récupérer les données. Sauvegardez le journal pendant l'exécution d'une sauvegarde complète.

Avant de pouvoir créer la première sauvegarde du journal, vous devez créer une sauvegarde complète, telle qu'une sauvegarde de base de données ou la première d'une série de sauvegardes de fichiers. La restauration d'une base de données à l'aide seulement de sauvegardes de fichiers peut être complexe. Par conséquent, nous vous recommandons de commencer par une sauvegarde de base de données complète dès que possible. Puis, sauvegardez le journal des transactions régulièrement. Vous pouvez ainsi réduire les risques de perte de travail mais aussi permettre la troncation du journal des transactions. En général, le journal des transactions est tronqué après chaque sauvegarde de journal conventionnelle.

Séquence de journaux de transactions consécutifs

Une séquence continue de sauvegarde de journaux s'appelle une séquence de journaux de transactions consécutifs. Une séquence de journaux de transactions consécutifs commence par une sauvegarde complète de la base de données. Généralement, une nouvelle séquence de journaux de transactions consécutifs ne démarre que lorsque la base de données est sauvegardée pour la première fois ou après que le mode de récupération simple est remplacé par le mode de récupération complète ou le mode de récupération utilisant les journaux de transactions. Si vous ne choisissez pas de remplacer les jeux de sauvegarde existants lors de la création d'une sauvegarde complète de base de données, la séquence de journaux de transactions consécutifs existante reste intacte. Grâce à la séquence de journaux de transactions consécutifs intacte, vous pouvez restaurer votre base de données à partir d'une sauvegarde complète de base de données du support de sauvegarde, suivie de toutes les sauvegardes de fichiers journaux suivantes jusqu'à votre point de récupération. Le point de récupération peut être la fin de la dernière sauvegarde de fichier journal ou un point de récupération spécifique dans chacune des sauvegardes de fichiers journaux.

Pour restaurer une base de données jusqu'au point d'échec, la séquence de journaux de transactions consécutifs doit être intacte. Autrement dit, la séquence ininterrompue des sauvegardes des journaux de transactions doit aller jusqu'au point de défaillance. Le point de commencement de cette séquence du journal dépend du type des sauvegardes de données que vous restez : base de données, partitions ou fichiers. Pour une sauvegarde partielle ou de base de données, la séquence des sauvegardes des journaux doit s'étendre à partir de la fin d'une sauvegarde partielle ou de base de données. Pour un jeu de sauvegardes de fichiers, la séquence des sauvegardes des journaux doit s'étendre à partir du début d'un jeu complet de sauvegardes de fichiers.

Restaurer des sauvegardes du journal

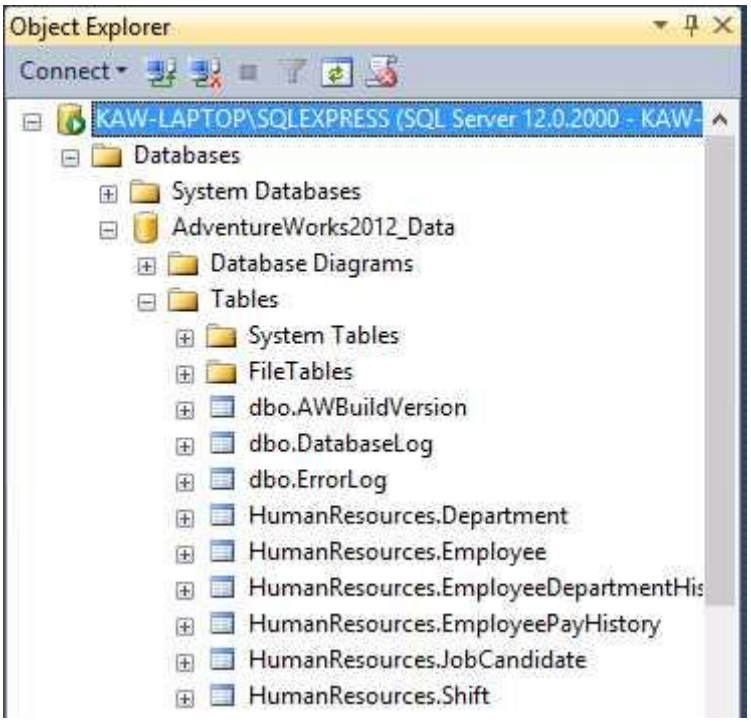
La restauration d'une sauvegarde de journal restaurer par progression les modifications enregistrées dans le journal des transactions, afin de recréer l'état exact de la base de données qui existait au début de la sauvegarde du journal. Lorsque vous restaurez une base de données, vous devez restaurer les sauvegardes des journaux créées à la suite de la sauvegarde complète de base de données que vous restaurez ou à partir de la première sauvegarde de fichiers que vous restaurez. En règle générale, vous devez restaurer une série de sauvegardes de journaux jusqu'au point de récupération, après avoir restauré les données les plus récentes ou une sauvegarde différentielle. Ensuite, vous récupérez la base de données. Cette opération restaure toutes les transactions qui n'étaient pas terminées au début de la récupération et place la base de données en ligne. Une fois la base de données récupérée, vous ne pourrez plus restaurer des sauvegardes.

3. Le dictionnaire de données du moteur choisi

Dans SQL Server, le dictionnaire de données est un ensemble de tables de base de données utilisées pour stocker des informations sur la définition d'une base de données. Le dictionnaire contient des informations sur les objets de base de données tels que les tables, les index, les colonnes, les types de données et les vues.
Le dictionnaire de données est utilisé par SQL Server pour exécuter des requêtes et est automatiquement mis à jour chaque fois que des objets sont ajoutés, supprimés ou modifiés dans la base de données.

SQL Server utilise le dictionnaire de base de données pour vérifier les instructions SQL. Lorsque vous exécutez une instruction SQL, le SGBD (Système de gestion de base de données) analyse l'instruction, puis détermine si les tables et les champs que vous référencez sont valides. Pour ce faire rapidement, il référence le dictionnaire de données.
En plus de tester la validité des instructions, SQL Server utilise le dictionnaire de données pour faciliter la génération du plan de requête et pour référencer les informations définissant la structure de la base de données.
Le dictionnaire de données devient un guide, en soi, que SQL Server utilise pour accéder à vos données. En termes simples, sans le dictionnaire de données, bien que SQL Server connaisse et comprenne le langage SQL, il ne connaîtra pas vos tables et colonnes de base de données ; par conséquent, il ne serait pas en mesure de les interroger.

Étant donné que le dictionnaire de données contient la définition de la base de données, c'est une très bonne ressource à utiliser pour obtenir des informations sur la base de données. Ce qui est vraiment cool, c'est que le dictionnaire de données est composé de tables et de vues SQL. Cela signifie que vous pouvez obtenir des informations sur la base de données via des requêtes !
Les dictionnaires de données sont utilisés par les concepteurs et les développeurs pour comprendre la structure de la base de données. Vous pouvez considérer le dictionnaire comme un document de référence à jour.
Les outils de conception tels que SQL Server Management Studio affichent des informations sur les bases de données via l'explorateur d'objets à l'aide du dictionnaire de données.

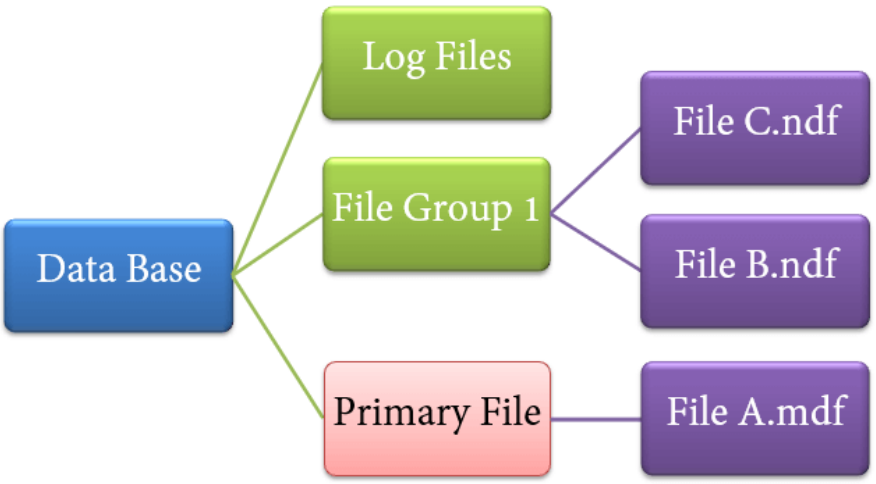


Les tables répertoriées ci-dessus ne sont pas connues par magie, l'explorateur d'objets a plutôt envoyé une requête au dictionnaire de données pour récupérer toutes les tables utilisateur.
Le dictionnaire de données est stocké dans une série de tables système. Bien que vous puissiez interroger directement ces tables, Microsoft se réserve le droit de modifier les tables système qui composent le dictionnaire de données. Pour cette raison, ils vous recommandent d'interroger les vues INFORMATION_SCHEMA plutôt que d'accéder directement aux tables.

Étant donné que vous pouvez interroger vous-même le dictionnaire de données, vous pouvez répondre à certaines questions qui nécessiteraient autrement beaucoup de recherche et de picorage dans l'explorateur d'objets. Par exemple, comment retrouver facilement toutes les tables et vues à l'aide de la colonne BusinessEntityID ? Sans le dictionnaire de données, vous devrez utiliser l'explorateur d'objets et ouvrir chaque table, afficher et parcourir les définitions de la colonne. Cependant, en utilisant le dictionnaire de données, cela peut être fait à l'aide d'une simple requête.

4. Organisation physique du SGBD

Les différents types de fichiers qui compose ce moteur et leur rôle



- **Fichier principal**

- Chaque base de données contient un fichier primaire.
- Cela stocke toutes les données importantes liées aux tables, vues, déclencheurs, etc.
- L'extension est . mdf généralement mais peut être de n'importe quelle extension.

- **Fichier secondaire**

- La base de données peut contenir ou non plusieurs fichiers secondaires.
- Ceci est facultatif et contient des données spécifiques à l'utilisateur.
- L'extension est . ndf généralement mais peut être de n'importe quelle extension.

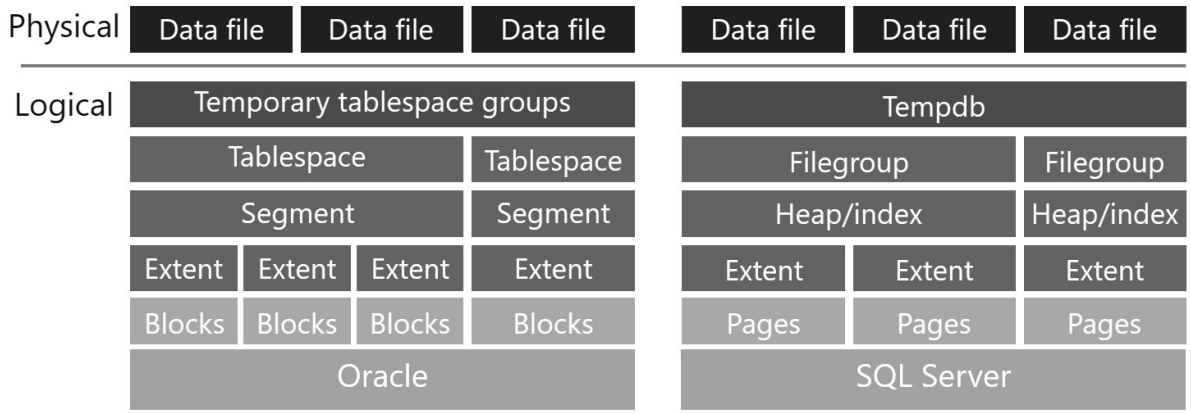
- **Fichier journal**

- Également connu sous le nom de journaux d'écriture anticipée.
- L'extension est . ldf
- Utilisé pour la gestion des transactions.
- Ceci est utilisé pour récupérer à partir de toutes les instances indésirables. Effectuez une tâche importante de restauration des transactions non validées.

https://www.guru99.com/sql-server-architecture.html

5. Organisation logique des données

Comment sont organisées de manière logique (tablespace / segments / block chez Oracle par exemple) les données des tables, les données des index, les données d'annulation et les données temporaires



Les nomenclatures diffèrent un peu, mais les fondamentaux ne changent jamais. Comment les blocs sont logiquement les mêmes que ceux que nous appelons les pages à l'intérieur de SQL Server. Le concept des tablespaces est similaire à la façon dont nous appelons des groupes de fichiers dans SQL Server.

Structure	Oracle	SQL Server
Smallest unit of logical storage	Block	Page
Block size	Variable	8 KB fixed
Storage allocation	Performed in multiple pages; are 'extents'	Performed in multiple pages; are 'extents'
Extent size	Variable	64 KB fixed
Segment	Any logical structure that is allocated storage	No equivalent structure

Les deux se ressemblent, lorsque les données sont persistantes, il y a des différences à comprendre. La différence la plus frappante ici concerne la façon dont SQL Server a une taille de bloc (page) fixe de 8 Ko qui, par rapport à Oracle, est variable.

6. Gestion de la concurrence d'accès

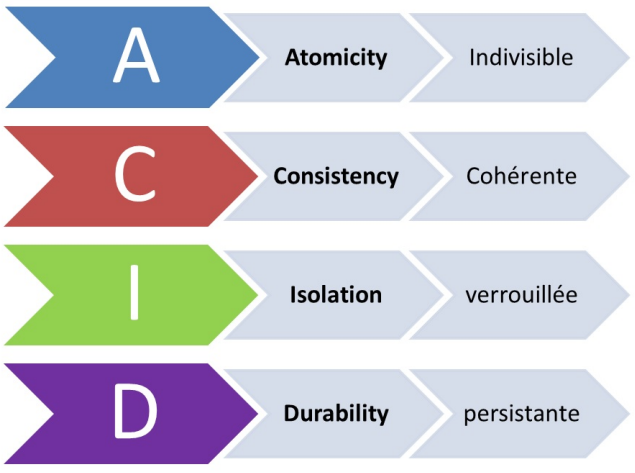
Une transaction est une suite d'opérations effectuées comme une seule unité logique de travail. Une unité logique de travail doit posséder quatre propriétés appelées **propriétés ACID** (Atomicité, Cohérence, Isolation et Durabilité), pour être considérée comme une transaction :

Atomicité
Une transaction doit être une unité de travail indivisible ; soit toutes les modifications de données sont effectuées, soit aucune ne l'est.

Cohérence
Lorsqu'elle est terminée, une transaction doit laisser les données dans un état cohérent. Dans une base de données relationnelle, toutes les règles doivent être appliquées aux modifications apportées par la transaction, afin de conserver l'intégrité de toutes les données. Toutes les structures de données internes, comme les index B-tree ou les listes à chaînage double, doivent être cohérentes à la fin de la transaction.

Isolement
Les modifications effectuées par des transactions concurrentes doivent être isolées transaction par transaction. Une transaction reconnaît les données dans l'état où elles se trouvaient avant d'être modifiées par une transaction simultanée, ou les reconnaît une fois que la deuxième transaction est terminée, mais ne reconnaît jamais un état intermédiaire. Cette propriété est nommée mise en série, car elle permet de recharger les données de départ et de répéter une suite de transactions dont le résultat sur les données sera identique à celui des transactions d'origine.

Durabilité
Lorsqu'une transaction durable est terminée, ses effets sur le système sont permanents. Les modifications sont conservées même en cas de défaillance du système. SQL Server 2014 (12.x) et versions ultérieures permettent les transactions durables retardées. Les transactions durables retardées sont validées avant de consigner l'enregistrement du journal des transactions sur le disque.



Les programmeurs SQL doivent concevoir des transactions dont les points de début et de fin permettent de maintenir la cohérence logique des données. La séquence de modifications des données qu'ils définissent doivent laisser les données dans un état cohérent par rapport aux règles d'entreprise définies par leur société.
Ces instructions de modification des données doivent par conséquent être contenues dans une seule transaction pour que le Moteur de base de données SQL Server puisse garantir l'intégrité physique de la transaction.

Un système de base de données d'entreprise, par exemple une instance de Moteur de base de données SQL Server, se doit de fournir des mécanismes permettant de garantir l'intégrité physique de chaque transaction. Le Moteur de base de données SQL Server fournit les éléments suivants :

- Des fonctionnalités de verrouillage permettant d'assurer l'isolement des transactions.

- Des fonctionnalités de consignation assurent la durabilité des transactions. Pour les transactions durables, l'enregistrement du journal est renforcé sur le disque avant les validations des transactions. Ainsi, en cas de défaillance du matériel serveur, du système d'exploitation ou de l'instance du Moteur de base de données SQL Server lui-même, l'instance utilise au redémarrage les journaux des transactions pour restaurer automatiquement toutes les transactions incomplètes jusqu'au moment de la défaillance du système. Les transactions durables retardées sont validées avant de renforcer l'enregistrement du journal des transactions sur le disque. Ces transactions peuvent être perdues en cas de défaillance du système avant que l'enregistrement du journal ne soit renforcé sur le disque.

- Des fonctionnalités de gestion des transactions qui assurent l'atomicité et la cohérence des transactions. Lorsqu'une transaction a débuté, elle doit se dérouler correctement jusqu'à la fin (validée), sans quoi l'instance du Moteur de base de données SQL Server annule toutes les modifications effectuées sur les données depuis le début de la transaction. Cette opération est appelée restauration d'une transaction, car elle retourne les données telles qu'elles étaient avant ces modifications.

Technique de gestion de la concurrence d'accès

Lorsque plusieurs utilisateurs accèdent à une ressource en même temps, on parle d'accès concurrentiel. L'accès concurrentiel aux données requiert certains mécanismes permettant de contrer les effets négatifs de la modification d'une ressource déjà en cours d'utilisation.

Effet des accès concurrentiels

Les utilisateurs qui modifient des données peuvent interférer avec d'autres utilisateurs en train de lire ou de modifier les mêmes données en même temps. On dit que ces utilisateurs accèdent aux données de manière concurrentielle. Si un système de stockage de données est dépourvu de contrôle des accès concurrentiels, les utilisateurs peuvent constater les effets secondaires suivants :

- **Mises à jour perdues**

Les mises à jour perdues se produisent lorsque deux transactions ou plus sélectionnent la même ligne, puis la mettent à jour en fonction de la valeur qui s'y trouvait à l'origine. Aucune transaction n'a connaissance des autres transactions. La dernière mise à jour écrase les mises à jour effectuées par les autres transactions, ce qui entraîne une perte de données.

Exemple : deux éditeurs font une copie électronique du même document. Chaque éditeur modifie son document et l'enregistre ensuite, en écrasant le document original. Le dernier éditeur à avoir enregistré le document écrase les modifications effectuées par l'autre éditeur. Le problème pourrait être évité en empêchant un éditeur d'accéder au fichier tant que l'autre éditeur n'a pas terminé et validé la transaction.

- **Dépendance non validée (lecture erronée)**

Une dépendance non validée se produit lorsqu'une deuxième transaction sélectionne une ligne qui est actuellement mise à jour par une autre transaction. La deuxième transaction lit les données qui n'ont pas encore été validées et qui peuvent être modifiées par la transaction de mise à jour de la ligne.

Supposons par exemple qu'un éditeur effectue des modifications dans un document électronique. Pendant les modifications, un second éditeur fait une copie du document comprenant toutes les modifications effectuées jusqu'alors et distribue ce dernier aux destinataires concernés. Le premier éditeur décide alors que les modifications effectuées sont incorrectes, les supprime et enregistre le document. Le document qui a été distribué comprend donc des modifications qui n'existent plus et devraient être traitées comme si elles n'avaient jamais existé. Le problème pourrait être évité en interdisant la lecture du document modifié tant que le premier éditeur n'a pas effectué l'enregistrement final des modifications et validé la transaction.

- **Analyse incohérente (lecture non reproductible)**

Une analyse incohérente se produit lorsqu'une deuxième transaction accède à la même ligne plusieurs fois et lit différentes données à chaque fois. Une analyse incohérente est similaire à une dépendance non validée en ce sens qu'une autre transaction change les données qu'une deuxième transaction est en train de lire. Cependant, dans une analyse incohérente, les données lues par la deuxième transaction sont validées par la transaction qui a effectué la modification. En outre, une analyse incohérente implique plusieurs lectures (deux ou plus) de la même ligne dont les informations sont systématiquement modifiées par une autre transaction, d'où l'expression de lecture non renouvelable.

Par exemple, un éditeur relit le même document deux fois, mais l'auteur réécrit le document entre les relectures. Lorsque l'éditeur relit le document pour la seconde fois, le document a changé. La relecture initiale n'est donc pas renouvelable. Ce problème pourrait être évité si l'auteur ne pouvait pas modifier le document tant que l'éditeur n'a pas terminé la dernière lecture.

- **Lectures fantômes**

Une lecture fantôme est une situation qui se produit lorsque deux requêtes identiques sont exécutées et que la collection de lignes retournée par la deuxième requête est différente. L'exemple suivant montre comment cela peut arriver. Supposons que les deux transactions ci-dessous s'exécutent en même temps. Les deux instructions **SELECT** dans la première transaction peuvent retourner des résultats différents parce que l'instruction **INSERT** dans la deuxième transaction modifie les données utilisées par les deux transactions.

```sql
--Transaction 1
BEGIN TRAN;
SELECT ID FROM dbo.employee
WHERE ID > 5 and ID < 10;
--The INSERT statement from the second transaction occurs here.
SELECT ID FROM dbo.employee
WHERE ID > 5 and ID < 10;
COMMIT;
```

```sql
--Transaction 2
BEGIN TRAN;
INSERT INTO dbo.employee
  (Id, Name) VALUES(6 ,'New');
COMMIT;
```

- **Lectures manquantes et en double provoquées par des mises à jour de ligne**

- Manquer une ligne mise à jour ou consulter une ligne mise à jour plusieurs fois

Les transactions qui s'exécutent au niveau READ UNCOMMITTED ne génèrent pas de verrous partagés pour empêcher d'autres transactions de modifier des données lues par la transaction en cours. Les transactions qui s'exécutent au niveau READ COMMITTED génèrent des verrous partagés, mais les verrous de ligne ou de page sont libérés une fois la ligne lue. Dans les deux cas, lorsque vous analysez un index, si un autre utilisateur modifie la colonne de clé d'index de la ligne pendant votre lecture, la ligne peut apparaître de nouveau si la modification apportée à la clé a déplacé la ligne à une position située en aval de votre analyse. De même, la ligne peut ne pas apparaître si la modification apportée à la clé a déplacé la ligne à une position dans l'index que vous aviez déjà lue. Pour éviter cela, utilisez l'indicateur SERIALIZABLE ou HOLDLOCK, ou le contrôle de version de ligne.

- Manquer une ou plusieurs lignes qui n'étaient pas la cible de la mise à jour

Quand vous utilisez READ UNCOMMITTED, si votre requête lit des lignes à l'aide d'une analyse d'ordre d'allocation (à l'aide de pages IAM), vous risquez de manquer des lignes si une autre transaction provoque un fractionnement de page. Cela ne peut pas se produire lorsque vous utilisez la lecture validée car un verrou de table est maintenu pendant un fractionnement de page et ne se produit pas si la table n'a pas d'index cluster, car les mises à jour ne provoquent pas de fractionnements de page.

Niveau d'isolation	Définition
Lecture non validée	Le niveau d'isolement le plus bas et suffisant pour s'assurer que les données physiquement corrompues ne sont pas lues. À ce niveau, les lectures de modifications sont autorisées. Ainsi, une transaction peut afficher les modifications qui ne sont pas encore validées apportées par d'autres transactions.
Lecture validée	Permet à une transaction de lire des données lues auparavant (non modifiées) par une autre transaction, sans attendre la fin de la première transaction. Le Moteur de base de données SQL Server conserve les verrous d'écriture (acquis sur les données sélectionnées) jusqu'à la fin de la transaction, mais les verrous le lecture sont libérés dès que l'opération SELECT est terminée. C'est le niveau par défaut du Moteur de base de données SQL Server.
Lecture renouvelée	Le Moteur de base de données SQL Server conserve les verrous de lecture et d'écriture acquis sur les données sélectionnées jusqu'à la fin de la transaction. Cependant, étant donné que les verrous d'étendus ne sont pas gérés, des lectures fantômes peuvent se produire.
Sérialisable	Niveau le plus élevé, dans lequel les transactions sont totalement isolées les unes des autres. Le Moteur de base de données SQL Server conserve les verrous de lecture et d'écriture acquis sur les données sélectionnées de façon à les libérer à la fin de la transaction. Les verrous d'étendues sont acquis lorsqu'une opération SELECT utilise une clause WHERE, plus particulièrement pour éviter les lectures fantômes. **Remarque :** Les opérations DDL et les transactions sur les tables répliquées peuvent échouer lorsque le niveau d'isolation sérialisable est demandé. Cela est dû au fait que les requêtes de réplication utilisent des indicateurs qui peuvent être incompatibles avec le niveau d'isolation sérialisable.

Le tableau suivant répertorie les effets secondaires de la concurrence provoqués par les différents niveaux d'isolation.

Niveau d'isolation	Lecture incorrecte	Lecture non renouvelable	Fantôme
Lecture non validée	Oui	Oui	Oui
Lecture validée	Non	Oui	Oui
Lecture renouvelée	Non	Non	Oui
Instantané	Non	Non	Non
Sérialisable	Non	Non	Non

7. Gestion des transactions distribuées

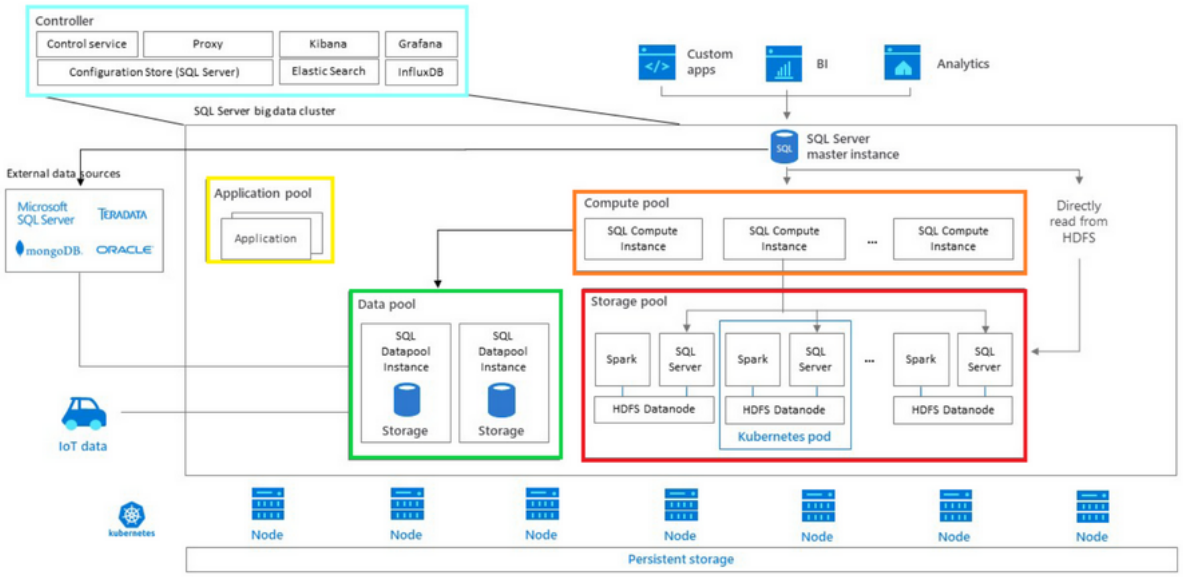
- **Architecture**

Les utilisateurs d' OLE DB Native Client, peuvent utiliser la méthode “ITransactionJoin :: JoinTransaction” pour participer à une transaction distribuée. Les transactions distribuées sont coordonnées par MS DTC (Microsoft Distributed Transaction Coordinator).

C’est grâce à des objets COM (Component Object Model de Microsoft), que les clients peuvent initialiser des transactions coordonnées et y participer via plusieurs connexions à différentes banques de données.

Le lancement d'une transaction se fait via l'interface MS DTC ITransactionDispenser, dont le membre BeginTransaction retourne une référence sur un objet de transaction distribuée. C'est cette même référence qui est passée au serveur fournisseur de SQL Server à l'aide de JoinTransaction.

La validation asynchrones sur les transactions distribuées et l'abandon sont pris en charge par MS DTC.



De plus, depuis la version 2019, SQL Server est une plateforme de données unifiée, qui, grâce à la nouvelle technologie Big Data Clusters, prend en charge l'apprentissage automatique, l'analyse de données, la conteneurisation des applications et son orchestration via Kubernetes et la virtualisation des données avec Azure.

Big Data Clusters (BDC) est une implémentation en grappes multiples de SQL Server orchestrée par Kubernetes, intégrant le framework de calcul distribué Apache Spark, le gestionnaire de clusters Hadoop YARN, le système de fichiers distribué de Hadoop [HDFS], Apache Hive, Apache Knox, Apache Ranger et Apache Livy ; pour proposer une plateforme unique qui peut prendre en charge le traitement des transactions en ligne OLTP, les données non structurées et même les besoins en Machine Learning.

Un Big Data Cluster se compose d'un **Contrôleur** (Controler), d'une Instance SQL Server Master (SQL Server master instance) et de 4 pools, **Le pool de calcul** (Compute pool), **Le pool de stockage** (Storage pool), **le pool de données** (Data pool) et le **pool d'application** (App pool).

- **Positionnement par rapport aux 12 règles de Date**

<u>Règle 1 : La règle de l'information *(The information rule)*</u>

Toutes les informations d'une base de données relationnelle sont représentées d'une seule et unique manière - par des valeurs dans des tables. Cette règle est une définition informelle d'une base de données relationnelle et indique que chaque élément de données que nous stockons de façon permanente dans une base de données se trouve dans une table. En général, SQL Server répond à cette règle, car nous ne pouvons pas stocker d'informations ailleurs que dans une table. Nous ne pouvons pas utiliser les variables dans ce code pour conserver des données, et elles sont donc limitées à un seul lot.

Règle 2 : La règle de garantie d'accès *(Guaranteed access rule)*

L'accès logique de chaque donnée (valeur atomique) est garanti par la combinaison du nom de la table, de la valeur de la clé primaire et du nom de la colonne. Cette règle souligne l'importance des clés primaires pour localiser les données dans la base de données. Le nom de la table permet de localiser la bonne table, le nom de la colonne permet de trouver la bonne colonne et la valeur de la clé primaire permet de trouver la ligne contenant un élément de données individuel intéressant. En d'autres termes, chaque élément (atomique) de données est accessible par la combinaison du nom de la table, de la valeur de la clé primaire et du nom de la colonne. Cette règle spécifie exactement la façon dont nous accédons aux données à l'aide d'un langage d'accès tel que Transact-SQL (T-SQL) dans SQL Server. En utilisant SQL, nous pouvons rechercher la valeur de la clé primaire (dont l'unicité est garantie, sur la base de la théorie relationnelle, tant qu'elle a été définie), et une fois que nous avons la ligne, les données sont accessibles par le nom de la colonne. Nous pouvons également accéder aux données par n'importe quelle colonne de la table, mais nous ne sommes pas toujours assurés de recevoir une ligne unique en retour.

Règle 3: La règle de traitement systématique des valeurs nulles (*Systematic Treatment of NULL Values)*

Les valeurs NULL sont prises en charge par le SGBD relationnel complet pour représenter les informations manquantes de manière systématique, indépendamment du type de données. Cette règle exige que le SGBDR prenne en charge un support distinct pour les valeurs NULL, quel que soit le type de données. Les NULL doivent se propager dans les opérations mathématiques ainsi que dans les opérations sur les chaînes de caractères.
NULL +<n'importe quoi> = NULL, la logique étant que NULL signifie "inconnu"
Le fait est que si on ajoute quelque chose de connu à quelque chose d'inconnu, on se sait toujours pas ce que nous avons, donc c'est toujours inconnu. Il existe quelques paramètres dans SQL Server qui peuvent personnaliser la façon dont les NULL sont traités. La plupart de ces paramètres existent à cause de certaines mauvaises pratiques qui étaient autorisées dans les premières versions de SQL Server.

Règle 4 : Catalogue dynamique en ligne basé sur le modèle relationnel

La description de la base de données est représentée au niveau logique de la même manière que les données ordinaires, de sorte que les utilisateurs autorisés peuvent appliquer à son interrogation le même langage relationnel que celui qu'ils appliquent aux données ordinaires.

Cette règle exige qu'une base de données relationnelle soit auto-décrite. En d'autres termes, la base de données doit contenir certaines tables système dont les colonnes décrivent la structure de la base de données elle-même, ou, de manière alternative, la description de la base de données est contenue dans les tables accessibles à l'utilisateur. Dans SQL Server, il existe la mise en œuvre des vues système "INFORMATION_SCHEMA". C'est un schéma qui comporte un ensemble de vues permettant d'examiner une grande partie des métadonnées des tables, des relations, des contraintes et même du code de la base de données. Ce sont les vues système qui ont remplacé les tables système utilisées depuis le début de SQL Server.

Règle 5 : règle du sous-langage des données complètes (Comprehensive Data Sublanguage Rule)

Un système relationnel peut supporter plusieurs langages et divers modes d'utilisation des terminaux. Toutefois, il doit exister au moins un langage dont les instructions sont exprimables, selon une syntaxe bien définie, sous forme de chaînes de caractères et dont la capacité à prendre en charge tous les éléments suivants est compréhensible :

- a. définition de données
- b. définition de vues
- c. manipulation des données (interactive et par programme)
- d. les contraintes d'intégrité
- e. l'autorisation
- f. les limites des transactions (début, fin et retour).

Cette règle impose l'existence d'un langage de base de données relationnel, tel que SQL, pour manipuler les données. Le langage SQL en tant que tel n'est pas spécifiquement requis. Le langage doit être capable de prendre en charge toutes les fonctions centrales d'un SGBD : création d'une base de données, extraction et saisie de données, mise en œuvre de la sécurité des bases de données, etc. T-SQL remplit cette fonction pour SQL Server et effectue toutes les tâches de définition et de manipulation des données nécessaires pour accéder aux données. SQL est un langage non procédural, en ce sens que vous ne spécifiez pas "comment" les choses se passent, ni même où. Vous posez simplement une question au serveur relationnel et il fait le travail.

Règle 6 : Règle de mise à jour des vues

Toutes les vues qui peuvent théoriquement être mises à jour le sont également par le système. Cette règle concerne les vues, qui sont des tables virtuelles utilisées pour donner aux différents utilisateurs d'une base de données des vues différentes de sa structure. C'est l'une des règles les plus difficiles à mettre en œuvre dans la pratique, et aucun produit commercial ne la satisfait pleinement aujourd'hui. Une vue peut théoriquement être mise à jour tant qu'elle est composée de colonnes qui correspondent directement à des colonnes de tables réelles. Dans SQL Server, les vues peuvent être mises à jour tant qu'on ne met pas à jour plus d'une seule table dans la déclaration ; on ne peut pas non plus mettre à jour un champ dérivé ou constant. SQL Server 2000 a également mis en place des déclencheurs INSTEAD OF que nous pouvons appliquer à une vue. Par conséquent, cette règle peut être techniquement respectée en utilisant des déclencheurs INSTEAD OF triggers, mais d'une manière qui peut être moins directe.

Règle 7 : Insertion, mise à jour et suppression de haut niveau

La capacité de traiter une relation de base ou une relation dérivée comme une seule opération s'applique non seulement à la récupération des données mais aussi à l'insertion, à la mise à jour et à la suppression des données. Cette règle souligne la nature orientée “ensemble” d'une base de données relationnelle. Elle exige que les lignes soient traitées comme des actifs dans les opérations d'insertion, de suppression et de mise à jour. La règle est conçue pour interdire les implémentations qui ne prennent en charge que la modification de la base de données ligne par ligne et par navigation. Le langage SQL couvre cela via les instructions INSERT, UPDATE et DELETE. Même le CLR ne permet pas d'accéder aux fichiers physiques où les données sont stockées, mais BCP contourne en quelque sorte ce problème. Comme toujours, il faut être très prudent lorsqu'on utilise les outils de bas niveau qui peuvent modifier les données sans passer par la syntaxe SQL typique, car ils peuvent ignorer les règles que nous avons établies, introduisant des incohérences dans les données.

Règle 8 : Indépendance physique des données

Les programmes d'application et les activités des terminaux restent logiquement inchangés lorsque des modifications sont apportées à la représentation des données ou aux méthodes d'accès. Les applications doivent continuer à fonctionner en utilisant la même syntaxe. Cette règle implique que la manière dont les données sont stockées physiquement doit être indépendante de la manière logique dont on y accède. Cela signifie que les utilisateurs ne doivent pas se préoccuper de la façon dont les données sont stockées ou de la façon dont on y accède. En fait, les utilisateurs des données doivent seulement être en mesure d'obtenir la définition de base des données dont ils ont besoin. Les autres choses qui ne devraient pas affecter la vision des données par l'utilisateur sont les suivantes :

L'ajout d'index : Les index déterminent la façon dont les données sont stockées, mais l'utilisateur, grâce à SQL, ne saura jamais que des index sont utilisés.

Changer le groupe de fichiers d'un objet : Le simple fait de déplacer une table vers un nouveau groupe de fichiers n'aura aucune incidence sur l'application. Vous accédez aux données de la même manière, quel que soit l'endroit où elles se trouvent physiquement.
Utilisation du partitionnement : Outre le déplacement de tables entières vers différents groupes de fichiers, vous pouvez déplacer des parties d'une table en utilisant des technologies de partitionnement pour répartir l'accès à différents sous-systèmes indépendants afin d'améliorer les performances.

Modifier le moteur de stockage : De temps à autre, Microsoft doit modifier le fonctionnement de SQL Server (notamment lors des mises à jour majeures de la version). Toutefois, les instructions SQL doivent sembler accéder aux données de la même manière que dans toute version précédente, mais plus rapidement. Microsoft a investi beaucoup de travail dans ce domaine, car SQL Server possède un moteur relationnel et un moteur de stockage distincts, et OLE DB est utilisé pour faire passer les données entre les deux.

Règle 9 : Indépendance logique des données

Les programmes d'application et les activités terminales restent logiquement indépendants lorsque des modifications de préservation de l'information, de quelque nature que ce soit, qui permettent théoriquement l'indépendance, sont apportées aux tables de base. Avec la règle 8, cette règle isole l'utilisateur ou le programme d'application de l'implémentation de bas niveau de la base de données, elles spécifient ensembles que les techniques d'accès ou de stockage spécifiques utilisées par le SGBD - et même les modifications de la structure des tables de la base de données - ne devraient pas affecter la capacité de l'utilisateur à travailler avec les données. Ainsi, si on ajoute une colonne à une table et si les tables sont divisées de manière à ne pas ajouter ou soustraire de colonnes, les programmes d'application qui appellent la base de données ne devraient pas être affectés.

Règle 10 : Indépendance d'intégrité

Les contraintes d'intégrité spécifiques à une base de données relationnelle particulière doivent pouvoir être définies dans le sous-langage des données relationnelles et être stockées dans le catalogue, et non dans les programmes d'application. La base de données doit supporter au minimum les deux contraintes d'intégrité suivantes :

- Intégrité des entités : Aucun composant d'une clé primaire n'est autorisé à avoir une valeur NULL.
- Intégrité référentielle : Pour chaque valeur de clé étrangère distincte non NULL dans une base de données relationnelle, il doit exister une valeur de clé primaire correspondante dans le même domaine.

Cette règle stipule que le langage de la base de données doit prendre en charge les contraintes d'intégrité qui limitent les données pouvant être saisies dans la base de données et les modifications qui peuvent y être apportées. En d'autres termes, le SGBD doit prendre en charge en interne la définition et l'application de l'intégrité des entités (clés primaires) et de l'intégrité référentielle (clés étrangères). Il faut que la base de données soit capable de mettre en œuvre des contraintes pour protéger les données contre les valeurs invalides et qu'une conception soignée de la base de données soit nécessaire pour garantir l'intégrité référentielle.

SQL Server 2008 fait un excellent travail en fournissant les outils nécessaires pour faire de cette règle une réalité. Nous pouvons protéger nos données contre les valeurs invalides dans la plupart des cas possibles en utilisant des contraintes et des déclencheurs.

Règle 11 : Indépendance de la distribution

Le sous-langage de manipulation des données d'un SGBD relationnel doit permettre aux programmes d'application et aux activités terminales de rester logiquement intacts, que les données soient physiquement centralisées ou distribuées. Cette règle signifie que le langage de la base de données doit être capable de manipuler des données situées sur d'autres systèmes informatiques.
En substance, nous devons être en mesure de répartir les données du SGBD sur plusieurs systèmes physiques sans que l'utilisateur s'en rende compte. SQL Server 2008 prend en charge les transactions distribuées entre les sources SQL Server, ainsi que d'autres types de sources utilisant le service Microsoft Distributed Transaction Coordinator. Une autre possibilité d'indépendance de la distribution est un groupe de serveurs de bases de données travaillant ensemble plus ou moins comme un seul. Avec chaque nouvelle version de SQL Server, la notion de serveurs de bases de données fédérés partageant la charge de manière transparente devient une réalité.

Règle 12 : Règle de non-subversion

Si un système relationnel possède ou supporte un langage de bas niveau (un seul enregistrement à la fois), ce langage de bas niveau ne peut pas être utilisé pour subvertir ou contourner les règles ou les contraintes d'intégrité exprimées dans le langage relationnel de plus haut niveau (plusieurs enregistrements à la fois).
Cette règle exige que les autres méthodes d'accès aux données ne puissent pas contourner les contraintes d'intégrité, ce qui signifie que les utilisateurs ne peuvent en aucun cas violer les règles de la base de données.

Dans les applications SQLServer 2008, cette règle est respectée, car il n'existe pas de méthodes permettant d'accéder aux données brutes et de modifier les valeurs autrement que par les méthodes prescrites par la base de données. Cependant, SQLServer 2008 enfreint cette règle à deux endroits :

- La copie en masse : Par défaut, vous pouvez utiliser les routines de copie en masse pour insérer des données dans la table directement et contourner les validations du serveur de base de données.
- Désactivation des contraintes et des déclencheurs : Il existe une syntaxe pour désactiver les contraintes et les déclencheurs, ce qui permet de contourner cette règle. Il est toujours bon de s'assurer que vous utilisez ces deux fonctionnalités avec précaution. Elles laissent des trous béants

dans l'intégrité de vos données, car elles permettent l'insertion de n'importe quelle valeur dans n'importe quelle colonne. Comme vous vous attendez à ce que les données soient protégées par la contrainte que vous avez appliquée, des erreurs de valeur de données peuvent se produire dans les programmes qui utilisent les données, sans les revalider au préalable.

- **Méthode pour garantir l'atomicité, la cohérence et la durabilité**

Les mécanismes de verrouillage et de contrôle de version de ligne utilisés par le Moteur de base de données SQL Server pour garantir l'intégrité physique de chaque transaction fonctionnent comme suit :

Atomicité
Une transaction doit être une unité de travail indivisible ; soit toutes les modifications de données sont effectuées, soit aucune ne l'est.

Cohérence
Lorsqu'elle est terminée, une transaction doit laisser les données dans un état cohérent. Dans une base de données relationnelle, toutes les règles doivent être appliquées aux modifications apportées par la transaction, afin de conserver l'intégrité de toutes les données. Toutes les structures de données internes, comme les index B-tree ou les listes à chaînage double, doivent être cohérentes à la fin de la transaction.

Durabilité
Lorsqu'une transaction durable est terminée, ses effets sur le système sont permanents. Les modifications sont conservées même en cas de défaillance du système. SQL Server 2014 (12.x) et versions ultérieures permettent les transactions durables retardées. Les transactions durables retardées sont validées avant de consigner l'enregistrement du journal des transactions sur le disque.

Les programmeurs SQL doivent concevoir des transactions dont les points de début et de fin permettent de maintenir la cohérence logique des données. La séquence de modifications des données qu'ils définissent doivent laisser les données dans un état cohérent par rapport aux règles d'entreprise définies par leur société. Ces instructions de modification des données doivent par conséquent être contenues dans une seule transaction pour que le Moteur de base de données SQL Server puisse garantir l'intégrité physique de la transaction.

Un système de base de données d'entreprise, par exemple une instance de Moteur de base de données SQL Server, se doit de fournir des mécanismes permettant de garantir l'intégrité physique de chaque transaction. Le Moteur de base de données SQL Server fournit les éléments suivants :

- Des fonctionnalités de verrouillage permettant d'assurer l'isolement des transactions.

- Des fonctionnalités de consignation assurent la durabilité des transactions. Pour les transactions durables, l'enregistrement du journal est renforcé sur le disque avant les validations des transactions. Ainsi, en cas de défaillance du matériel serveur, du système d'exploitation ou de l'instance du Moteur de base de données SQL Server lui-même, l'instance utilisé au redémarrage les journaux des transactions pour restaurer automatiquement toutes les transactions incomplètes jusqu'au moment de la défaillance du système. Les transactions durables retardées sont validées avant de renforcer l'enregistrement du journal des transactions sur le disque. Ces transactions peuvent être perdues en cas de défaillance du système avant que l'enregistrement du journal ne soit

renforcé sur le disque.

- Des fonctionnalités de gestion des transactions qui assurent l'atomicité et la cohérence des transactions. Lorsqu'une transaction a débuté, elle doit se dérouler correctement jusqu'à la fin (validée), sans quoi l'instance du Moteur de base de données SQL Server annule toutes les modifications effectuées sur les données depuis le début de la transaction. Cette opération est appelée restauration d'une transaction, car elle retourne les données telles qu'elles étaient avant ces modifications.

- **Traitement des pannes :** *Récupération d'une transaction*

Une transaction débute par un begin transaction et se termine par un commit ou un rollback. L'opération commit détermine le point ou la base de données est de nouveau cohérente. L'opération rollback annule toutes les opérations et retourne la base de données dans l'état où elle était au moment du begin transaction, donc du dernier commit.

Une transaction n'est pas uniquement une unité logique de traitement des données, c'est aussi une unité de récupération.Après qu'une transaction ait terminé avec succès (commit) le SGBDR garantit que les changements seront permanents dans la BD.

Cela ne veut pas dire, cependant, que les changements ont été écrits sur le disque dans le fichier physique de la BD. Ils peuvent être encore seulement dans la mémoire de l'ordinateur.

Si après le commit et avant que les changements soient écrits sur le disque, une panne électrique vient tout effacer le contenu de la mémoire et en même temps les changements tout juste 'comités', alors, le SGBDR sera quand même capable, au redémarrage, de poursuivre la mise à jour en récupérant la transaction des journaux.

Cela est possible à cause d'une règle qui stipule que les journaux sont physiquement sauvegardés sur le disque avant que le commit complete. Cette double sauvegarde ou redondance des données permet de récupérer non seulement une transaction, mais une BD complète.

Au moment d'une panne électrique ou d'une panne d'ordinateur, le contenu de la mémoire est perdu. L'état des transactions en cours est perdu. Les transactions complétées, mais non écrites sont disponibles dans les journaux. Au moment du redémarrage du SGBDR, toutes les transactions qui n'ont pas complété seront annulées. Celles qui n'ont pas été sauvegardées dans la BD seront rejouées à partir des journaux.

À intervalle régulier le SGBDR sauvegarde le contenu de ses structures de données en mémoire dans le fichier physique de la BD. Au même moment, un enregistrement 'CheckPoint' est ajouté au journal indiquant que toutes les transactions complétées avant le CP sont contenues dans la BD sur le disque. Pour déterminer quelles transactions seront annulées et quelles transactions seront rejouées, le SGBDR utilise cet enregistrement CP dans le journal.

8. Gestion de la reprise sur panne

- **Les techniques d'annulation**

Le Buffer Manager garantit que le journal des transactions est écrit avant que les modifications apportées à la base de données ne soient écrites. (Cela s'appelle la journalisation à écriture anticipée.) Cette garantie est possible car SQL Server effectue le suivi de sa position actuelle dans le journal au moyen du LSN. Chaque fois qu'une page est modifiée, le LSN correspondant à l'entrée de journal pour ce changement est écrit dans l'en-tête de la page de données. Les pages sales peuvent être écrites sur le disque

uniquement lorsque le LSN sur la page est inférieur ou égal au LSN du dernier enregistrement écrit dans le journal. Le Buffer Manager garantit également que les pages de journal sont écrites dans un ordre spécifique, indiquant clairement quels blocs de journal doivent être traités après une défaillance du système, quel que soit le moment où la défaillance s'est produite.

Les enregistrements de journal pour une transaction sont écrits sur le disque avant que l'accusé de réception de validation ne soit envoyé au processus client, mais les données réellement modifiées peuvent ne pas avoir été physiquement écrites sur les pages de données. Bien que les écritures dans le journal soient asynchrones, au moment de la validation, le thread doit attendre que les écritures se terminent au point d'écrire l'enregistrement de validation dans le journal de la transaction. (SQL Server doit attendre que l'enregistrement de validation soit écrit afin qu'il sache que les enregistrements de journal pertinents sont en toute sécurité sur le disque.) Les écritures dans les pages de données sont complètement asynchrones. Autrement dit, les écritures sur les pages de données doivent uniquement être publiées sur le système d'exploitation et SQL Server peut vérifier ultérieurement qu'elles ont été effectuées. Ils ne doivent pas être complétés immédiatement car le journal contient toutes les informations nécessaires pour refaire le travail, même en cas de panne de courant ou de panne du système avant la fin de l'écriture. Le système serait beaucoup plus lent s'il devait attendre la fin de chaque demande d'E / S avant de continuer.

- **La journalisation**

La journalisation consiste à délimiter le début et la fin de chaque transaction (et les points de sauvegarde, si une transaction les utilise). Entre les démarcations de début et de fin se trouvent des informations sur les modifications apportées aux données. Ces informations peuvent prendre la forme des données réelles «avant et après», ou elles peuvent faire référence à l'opération effectuée afin que ces valeurs puissent être dérivées. La fin d'une transaction typique est marquée par un enregistrement de validation, qui indique que la transaction doit être reflétée dans les fichiers de données de la base de données ou être refaite si nécessaire. Une transaction abandonnée pendant l'exécution normale (pas le redémarrage du système) en raison d'une restauration explicite ou de quelque chose comme une erreur de ressource (par exemple, une erreur de mémoire insuffisante) annule en fait l'opération en appliquant des modifications qui annulent les modifications de données d'origine.

Comme mentionné précédemment, les deux types de récupération ont pour objectif de s'assurer que le journal et les données concordent.

Une **récupération de redémarrage** s'exécute à chaque démarrage de SQL Server. Le processus s'exécute sur chaque base de données car chaque base de données possède son propre journal de transactions. Votre journal des erreurs SQL Server signale la progression de la récupération du redémarrage et, pour chaque base de données, le journal des erreurs vous indique le nombre de transactions annulées et le nombre de transactions annulées. Ce

type de récupération est parfois appelé récupération après incident, car une panne ou un autre arrêt inattendu du service SQL Server nécessite l'exécution du processus de récupération lorsque le service est redémarré. Si le service a été arrêté proprement sans aucune transaction ouverte dans aucune base de données, seule une récupération minimale est nécessaire au redémarrage du système. Dans SQL Server 2012, la récupération de redémarrage peut être exécutée sur plusieurs bases de données en parallèle,

L'autre type de restauration, **restauration de restauration** (ou restauration de média), est exécuté à la demande lorsqu'une opération de restauration est exécutée. Ce processus garantit que toutes les transactions validées dans la sauvegarde du journal des transactions sont reflétées dans les données et qu'aucune transaction incomplète n'apparaît dans les données. La restauration de la restauration est traitée plus en détail plus loin dans ce chapitre.

Le journal des transactions prend en charge les opérations suivantes :

- ➢ Récupération des transactions individuelles.
- ➢ Récupération de toutes les transactions incomplètes au démarrage de SQL Server
- ➢ Restauration par progression d'une base de données, d'un fichier, d'un groupe de fichiers ou d'une page jusqu'au point de défaillance
- ➢ Prise en charge de la réplication transactionnelle
- ➢ Prise en charge de la haute disponibilité et de solutions de récupération d'urgence : Groupes de disponibilité Always On, mise en miroir de bases de données et copie des journaux de transaction.

- **Les sauvegardes**

Les deux types de récupération doivent gérer deux situations: lorsque les transactions sont enregistrées comme validées dans le journal mais pas encore écrites dans les fichiers de données, et lorsque les modifications apportées aux fichiers de données ne correspondent pas aux transactions validées. Ces deux situations peuvent se produire car les enregistrements de journal validés sont écrits dans les fichiers journaux sur le disque chaque fois qu'une transaction est validée. Les pages de données modifiées sont écrites dans les fichiers de données sur le disque de manière complètement asynchrone, chaque fois qu'un point de contrôle se produit dans une base de données. Comme mentionné au chapitre 1les pages de données peuvent également être écrites sur le disque à d'autres moments, mais les opérations de point de contrôle qui se produisent régulièrement donnent à SQL Server un point auquel toutes les pages modifiées (ou modifiées) sont connues pour avoir été écrites sur le disque. Les opérations de point de contrôle écrivent également des enregistrements de journal des transactions en cours sur le disque, car les enregistrements de journal mis en cache sont également considérés comme sales.

Si le service SQL Server s'arrête après la validation d'une transaction, mais avant que les données ne soient écrites dans les pages de données, lorsque SQL Server démarre et exécute la récupération, la transaction doit être reportée. SQL Server refait essentiellement la transaction en réappliquant les modifications indiquées dans le journal des transactions. Toutes les transactions qui doivent être refaites sont traitées en premier (même si certaines

d'entre elles devront peut-être être annulées plus tard au cours de la phase suivante). C'est ce qu'on appelle la phase de rétablissement de la récupération.

Si un point de contrôle se produit avant qu'une transaction ne soit validée, il écrit les modifications non validées sur le disque. Si le service SQL Server s'arrête ensuite avant la validation, le processus de récupération trouve les modifications pour les transactions non validées dans les fichiers de données et doit annuler la transaction en annulant les modifications reflétées dans le journal des transactions. La restauration de toutes les transactions incomplètes s'appelle la phase d'annulation de la récupération.

Reprise

Les pannes d'un SGBD peuvent être classées en deux catégories :

- **panne d'ordinateur**, par exemple à la suite d'une coupure de courant : les données enregistrées en mémoire centrale sont perdues, mais pas celles enregistrées sur disque ;
- **panne de disque** : une partie ou toutes les données enregistrées sur ce disque sont perdues.

Gestion de la reprise

La reprise à **chaud**, après un abandon de transaction ou une panne d'ordinateur, peut être réalisée automatiquement et rapidement, en s'appuyant sur la tenue d'un journal qui garde en mémoire tous les événements d'une transaction.
Le journal de transaction est un fichier qui contient une suite d'enregistrements dont chacun décrit un événement d'une transaction.
Un journal est enregistré sur une **mémoire stable**, c'est-à-dire une mémoire qui théoriquement ne peut pas être détruite.
Si nécessaire, le journal est dupliqué en plusieurs exemplaires (par exemple, sur un disque miroir).
Les enregistrements sont écrits à partir de la fin du journal (mode append) et donc des plus anciens aux plus récents.

La reprise à **froid**, après une panne de disque, est plus longue à mettre en œuvre. Elle nécessite de réaliser des copies régulières de la BD et un archivage des mises à jour entre deux copies.

9. Techniques d'indexation

En principe, lorsqu'une requête SQL est envoyée au SGBD, celui-ci établit un plan d'exécution. Le module se charge d'établir un plan d'exécution s'appelle Optimizer.

Le fonctionnement de l'Optimizer globalement similaire pour l'ensemble des SGBDs (Oracle et SQL Server), en utilisant les étapes suivantes :

1. Validation syntaxique
2. Validation sémantique
3. Utilisation éventuelle d'un plan précédemment produit
4. Réécriture/Simplification de la requête
5. Exploration des chemins d'accès et estimation des coûts.
6. Désignation du chemin le moins coûteux, génération du plan d'exécution et mise en cache de ce dernier.

Les différentes techniques d'indexation :

Un index est un objet de la base de données permettant d'accélérer l'accès aux données. Le principe de recherche dans un index se fait un peu comme dans un B-Arbre un arbre parfaitement équilibré. Le rôle d'un index est d'accélérer la recherche d'information (lors d'un SELECT) dans une base une base de données.

1. Hachage

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.

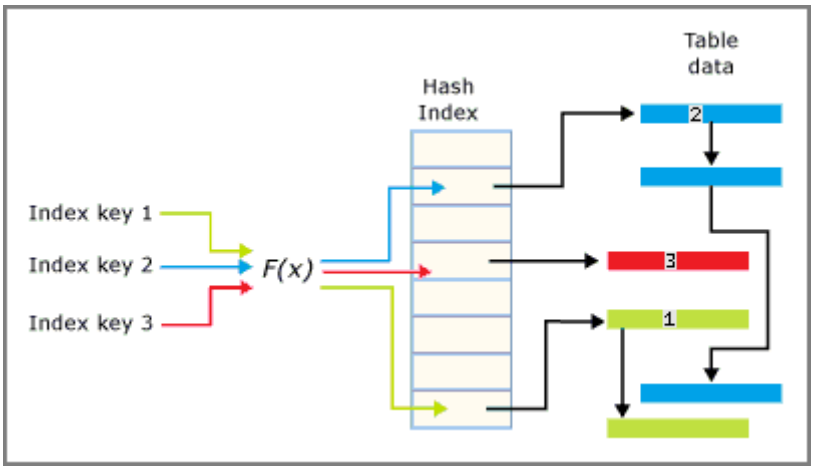
Un index de hachage se compose d'un tableau de pointeurs, chaque élément du tableau constituant un compartiment de hachage.

Chaque compartiment comprend 8 octets, qui sont utilisés pour stocker l'adresse mémoire d'une liste de liens d'entrées d'index.
Chaque entrée correspond à une valeur pour une clé d'index, plus l'adresse de la ligne associée dans la table mémoire optimisée sous-jacente.
Chaque entrée pointe vers l'entrée suivante dans une liste de liens d'entrées, toutes chaînées au compartiment actuel.
Le nombre de compartiments doit être spécifié au moment de la définition de l'index :

Plus le rapport entre les compartiments et les lignes de table ou les valeurs distinctes est faible, plus la liste moyenne de liens de compartiments sera longue.
Les listes de liens courtes sont plus rapides que les listes de liens longues.
Un index de hachage peut contenir 1 073 741 824 compartiments au maximum

La fonction de hachage est appliquée aux colonnes de clés d'index. Son résultat détermine le compartiment auquel ces clés sont mappées. Chaque compartiment a un pointeur vers les lignes dont les valeurs de clés hachées sont mappées à ce compartiment.La fonction de hachage est appliquée aux colonnes de clés d'index. Son résultat détermine le compartiment auquel ces clés sont mappées. Chaque compartiment a un pointeur vers les lignes dont les valeurs de clés hachées sont mappées à ce compartiment.

L'interaction entre l'index de hachage et les compartiments est résumée dans l'image ci-dessous :



Configuration du nombre de compartiments d'index de hachage :

Le nombre de compartiments d'index de hachage est spécifié au moment de la création de l'index et peut être modifié à l'aide de la syntaxe **ALTER TABLE...ALTER INDEX REBUILD.**

Dans la plupart des cas, le nombre de compartiments doit être compris entre 1 et 2 fois le nombre de valeurs distinctes dans la clé d'index.
Vous ne pouvez pas toujours prédire le nombre exact de valeurs qu'une clé d'index donnée a ou aura. Les performances sont en général encore bonnes si la valeur BUCKET_COUNT est inférieure ou égale à 10 fois le nombre réel de valeurs de clés et il est généralement préférable de surestimer que de sous-estimer.

Un trop petit nombre de compartiments présente les inconvénients suivants :

- Davantage de collisions de hachage de valeurs de clés distinctes.
- Chaque valeur distincte est forcée de partager le même compartiment avec une autre valeur distincte.
- La longueur de chaîne moyenne par compartiment augmente.
- Plus la chaîne de compartiment est longue, plus les recherches d'égalité sont lentes dans l'index.

Un trop grand nombre de compartiments présente les inconvénients suivants :

- Un nombre de compartiments trop élevé peut entraîner plus de compartiments vides.
- Des compartiments vides affectent les performances des analyses d'index complètes. Si celles-ci sont effectuées régulièrement, envisagez de choisir un nombre de compartiments proche du nombre de valeurs de clés distinctes.
- Des compartiments vides utilisent de la mémoire, même si chaque compartiment utilise seulement 8 octets.

Déclaration :

Un index de hachage peut exister uniquement sur une table optimisée en mémoire. Il ne peut pas exister sur une table sur disque.

Un index de hachage peut être déclaré comme :

- UNIQUE, ou être défini comme non unique par défaut.
- NONCLUSTERED, qui est la valeur par défaut.

Le code suivant est un exemple de la syntaxe appropriée pour créer un index de hachage, en dehors de l'instruction CREATE TABLE :

```
ALTER TABLE MyTable_memop
ADD INDEX ix_hash_Column2 UNIQUE
HASH (Column2) WITH (BUCKET_COUNT = 64);
```

2. 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.

Pour créer un index cluster :

1. Dans l' Explorateur d'objets, connectez-vous à une instance du Moteur de base de données.
2. Dans la barre d'outils standard, cliquez sur Nouvelle requête.
3. Copiez et collez l'exemple suivant dans la fenêtre de requête, puis cliquez sur Exécuter.

```
USE AdventureWorks2012;
GO
-- Create a new table with three columns.
CREATE TABLE dbo.TestTable
    (TestCol1 int NOT NULL,
     TestCol2 nchar(10) NULL,
     TestCol3 nvarchar(50) NULL);
GO
-- Create a clustered index called IX_TestTable_TestCol1
-- on the dbo.TestTable table using the TestCol1 column.
CREATE CLUSTERED INDEX IX_TestTable_TestCol1
    ON dbo.TestTable (TestCol1);
GO
```

Indications pour la conception d'index cluster :

Les index cluster trient et stockent les lignes de données de la table en fonction de leurs valeurs de clé. Il n'y a qu'un index cluster par table car les lignes de données ne peuvent être triées que dans un seul ordre. À quelques exceptions près, toutes les tables doivent avoir un index cluster défini sur la ou les colonnes présentant les caractéristiques suivantes :

- utilisables pour les requêtes fréquemment utilisées ;
- utilisables dans les requêtes de plage ;
- assurant un niveau élevé d'unicité

Lorsque vous créez une contrainte PRIMARY KEY, un index unique sur la ou les colonnes est automatiquement créé. Par défaut, cet index est cluster ; toutefois, vous pouvez spécifier un index non-cluster lorsque vous créez la contrainte.

Si l'index cluster n'est pas créé avec la propriété UNIQUE, le Moteur de base de données ajoute automatiquement une colonne d'indicateur d'unicité de quatre octets à la table. Si nécessaire, le

Moteur de base de données ajoute automatiquement une valeur d'indicateur d'unicité une ligne pour que chaque clé soit unique. Cette colonne et ses valeurs sont utilisées en interne et ne sont ni affichables, ni accessibles par les utilisateurs.

Architecture des index cluster :

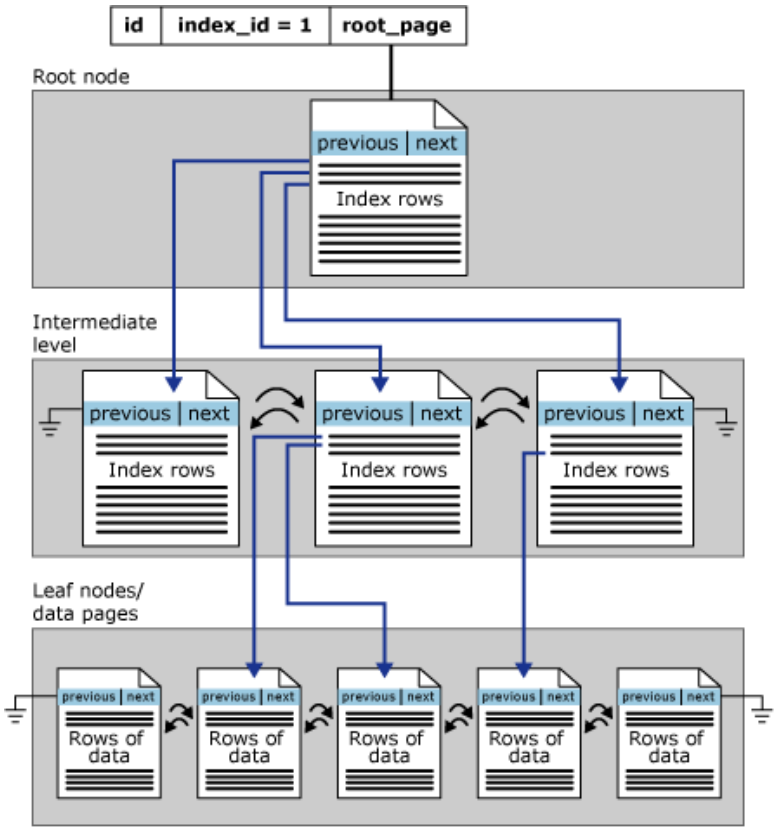
Dans SQL Server, les index sont organisés selon une arborescence binaire (B-tree). Chaque page d'une arborescence binaire d'index s'appelle un nœud d'index. Le nœud supérieur d'une arborescence binaire est le nœud racine. Les nœuds du niveau inférieur de l'index sont appelés les nœuds feuille. Tous les niveaux d'index situés entre la racine et les nœuds feuille s'appellent des niveaux intermédiaires. Dans un index cluster, les nœuds feuille contiennent les pages de données de la table sous-jacente. Les nœuds racine et de niveau intermédiaire contiennent les pages d'index dans lesquelles se trouvent les lignes d'index. Chaque ligne d'index contient une valeur de clé et un pointeur vers une page de niveau intermédiaire dans l'arborescence binaire ou vers une ligne de données dans le niveau feuille de l'index. Les pages de chaque niveau de l'index sont liées dans une liste à double liaison.

Un index cluster a une ligne dans sys.partitions, où index_id = 1 pour chaque partition utilisée par celui-ci. Par défaut, un index cluster possède une seule partition. Lorsqu'un index cluster détient plusieurs partitions, chacune d'elles possède une structure d'arborescence binaire qui contient ses données. Par exemple, si un index cluster possède quatre partitions, il existe quatre structures d'arborescence binaire, à raison d'une dans chaque partition.

Suivant les types de données de l'index cluster, chaque structure d'index cluster possède une ou plusieurs unités d'allocation pour le stockage et la gestion des données d'une partition spécifique. Au minimum, chaque index cluster détient une unité d'allocation IN_ROW_DATA par partition. L'index cluster possède également une unité d'allocation LOB_DATA par partition s'il contient des colonnes LOB (Large Object). De plus, il a une unité d'allocation ROW_OVERFLOW_DATA par partition s'il contient des colonnes de longueur variable qui dépassent la limite de taille de ligne de 8 060 octets.

Les pages de la chaîne de données et les lignes qu'elles rassemblent sont organisées en fonction de la valeur de la clé d'index cluster. Toutes les insertions sont faites à l'endroit où la valeur de clé de la ligne insérée correspond parfaitement à la séquence de tri parmi les lignes existantes.

L'illustration suivante montre la structure d'un index cluster dans une partition unique :



3. **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

Pour créer un index non-cluster sur une table :

1. Dans l' Explorateur d'objets, connectez-vous à une instance du Moteur de base de données.
2. Dans la barre d'outils standard, cliquez sur Nouvelle requête.
3. Copiez et collez l'exemple suivant dans la fenêtre de requête, puis cliquez sur Exécuter.

```
USE AdventureWorks2012;
GO
-- Find an existing index named IX_ProductVendor_VendorID and delete it if found.
IF EXISTS (SELECT name FROM sys.indexes
            WHERE name = N'IX_ProductVendor_VendorID')
    DROP INDEX IX_ProductVendor_VendorID ON Purchasing.ProductVendor;
GO
-- Create a nonclustered index called IX_ProductVendor_VendorID
-- on the Purchasing.ProductVendor table using the BusinessEntityID column.
CREATE NONCLUSTERED INDEX IX_ProductVendor_VendorID
    ON Purchasing.ProductVendor (BusinessEntityID);
GO
```

4. **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.

Indications pour la conception d'index non-cluster :

Un index non-cluster contient les valeurs de clé d'index et les localisateurs de ligne qui pointent vers l'emplacement de stockage des données de table. Vous pouvez créer plusieurs index non cluster sur une table ou une vue indexée. Les index non-cluster doivent, en principe, améliorer les performances des requêtes fréquemment utilisées qui ne sont pas couvertes par l'index cluster.

De la même manière que vous utilisez un index dans un livre, l'optimiseur de requête recherche une valeur de données en examinant l'index non-cluster afin de trouver l'emplacement qu'occupe la valeur dans la table, puis récupère directement les données à partir de cet emplacement. C'est pour cette raison que les index non cluster constituent une solution idéale pour les requêtes à correspondance exacte ; l'index contient en effet des entrées décrivant l'emplacement exact qu'occupent dans la table les valeurs de données recherchées dans les requêtes. Par exemple, pour interroger la table HumanResources. Employee pour tous les employés qui réfèrent à un responsable spécifique, l'optimiseur de requête peut utiliser l'index non cluster IX_Employee_ManagerID; sa colonne clé est ManagerID . L'optimiseur de requête recherche rapidement toutes les entrées de l'index qui correspondent à la valeur ManagerIDspécifiée. Chaque entrée d'index pointe vers la page et la ligne exactes de la table ou de l'index cluster contenant les données correspondantes. Après avoir trouvé toutes les entrées dans l'index, l'optimiseur de requête peut accéder directement à la page et à la ligne exactes pour récupérer les données.

Architecture des index non cluster :

Les index non-cluster possèdent la même structure arborescente binaire que les index cluster, à ces différences près :

- Les lignes de données de la table sous-jacente ne sont pas triées et stockées dans l'ordre des clés non cluster.
- Le niveau feuille d’un index non cluster n’est pas constitué de pages de données, mais de pages d’index.

Dans les lignes des index non-cluster, le localisateur est soit un pointeur vers une ligne, soit une clé d'index cluster :

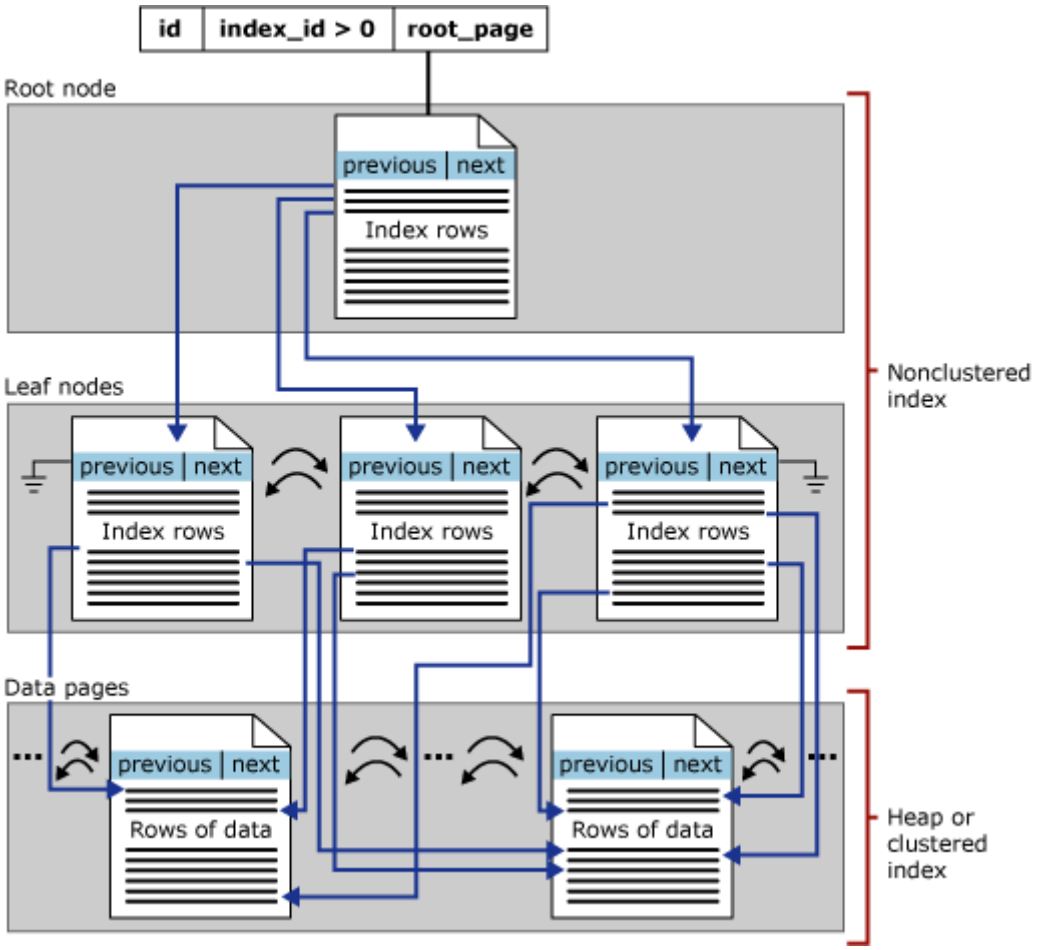
- Si la table est un segment de mémoire (dépourvue d'index cluster), le localisateur de ligne est un pointeur vers la ligne. Le pointeur est construit à partir de l'ID du fichier, du numéro de la page et du numéro de ligne dans la page. Le pointeur complet est appelé une ID de ligne (RID).
- Si la table a un index cluster, ou si l'index est sur une vue indexée, le localisateur de ligne est la clé d'index cluster pour la ligne.

Les index non-cluster comprennent une ligne dans sys.partitions où index_id >1 pour chaque partition utilisée par l’index. Par défaut, un index non-cluster contient une seule partition. Lorsqu'un

index non-cluster comprend plusieurs partitions, chaque partition a une structure arborescente binaire qui contient les lignes d'index correspondantes. Par exemple, si un index non-cluster a quatre partitions, il y a quatre arborescences binaires, une dans chaque partition.

En fonction des types de données de l'index non-cluster, chaque structure d'index non-cluster aura une ou plusieurs unités d'allocation dans lesquelles stocker et gérer les données d'une partition spécifique. Chaque index non-cluster aura au minimum une unité d'allocation IN_ROW_DATA par partition pour stocker les pages de l'arborescence binaire (B-tree) de l'index. L'index non-cluster aura également une unité d'allocation LOB_DATA par partition s'il contient des colonnes LOB (Large Object). Il aura par ailleurs une unité d'allocation ROW_OVERFLOW_DATA par partition s'il contient des colonnes de longueur variable dont les lignes dépassent la taille limite de 8 060 octets.

L'illustration suivante montre la structure d'un index non-cluster avec une seule partition :



5. Unique

Un index unique garantit que la clé de l'index ne contient pas de valeur dupliquée et que toute ligne de la table ou de la vue est en quelque sorte unique.

L'unicité peut être une propriété à la fois des index cluster et non cluster.

Pour créer un index unique sur une table :

1. Dans l' Explorateur d'objets, connectez-vous à une instance du Moteur de base de données.
2. Dans la barre d'outils standard, cliquez sur Nouvelle requête.
3. Copiez et collez l'exemple suivant dans la fenêtre de requête, puis cliquez sur Exécuter.

```
USE AdventureWorks2012;
GO
-- Find an existing index named AK_UnitMeasure_Name and delete it if found
IF EXISTS (SELECT name from sys.indexes
           WHERE name = N'AK_UnitMeasure_Name')
   DROP INDEX AK_UnitMeasure_Name ON Production.UnitMeasure;
GO
-- Create a unique index called AK_UnitMeasure_Name
-- on the Production.UnitMeasure table using the Name column.
CREATE UNIQUE INDEX AK_UnitMeasure_Name
   ON Production.UnitMeasure (Name);
GO
```

Les index uniques présentent les avantages suivants :

- L'intégrité des données des colonnes définies est garantie.
-
- L'optimiseur de requête dispose d'informations utiles supplémentaires.

Lorsque vous créez une contrainte PRIMARY KEY ou UNIQUE, vous créez automatiquement un index unique sur les colonnes spécifiées. Il n'y a pas de différences significatives entre la création d'une contrainte UNIQUE et la création d'un index unique indépendant d'une contrainte. La validation des données se produit de la même manière et l'optimiseur de requête considère un index unique de la même façon, qu'il soit créé par une contrainte ou manuellement. Toutefois, vous devez créer une contrainte UNIQUE ou PRIMARY KEY sur la colonne lorsque votre objectif est de préserver l'intégrité des données. Cette opération met en évidence la finalité de l'index.

6. **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.

7. **Index à colonnes incluses**

Index non cluster qui est étendu pour inclure des colonnes non clés en plus des colonnes clés.

8. **Index sur les colonnes calculées**

Index sur une colonne dérivée de la valeur d'une ou de plusieurs autres colonnes, ou certaines entrées déterministes

9. **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.

10. **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.

11. **XML**

Représentation fragmentée et permanente des objets binaires XML de taille importante (BLOB) dans la colonne de type de données xml.

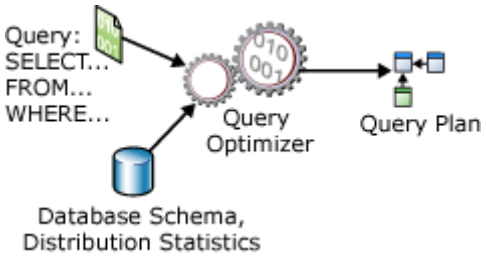
12. **Texte intégral**

Type spécial d'index fonctionnel par jeton qui est construit et géré par le Moteur d'indexation et de recherche en texte intégral Microsoft pour SQL Server. Il permet de prendre en charge efficacement toute recherche de mots sophistiqués dans des chaînes de données caractères.

10. Optimiseur de requêtes

Une instruction SELECT est non procédurale ; elle ne précise pas les étapes exactes à suivre par le serveur de base de données pour extraire les données demandées. Cela signifie que le serveur de base de données doit analyser l'instruction afin de déterminer la manière la plus efficace d'extraire les données demandées. Cette opération est nommée optimisation de l'instruction SELECT . Le composant qui s'en charge est l'optimiseur de requête. L'entrée de l'optimiseur de requête est composée de la requête, du schéma de base de données (définitions des tables et des index) et de ses statistiques de base de données. La sortie de l'optimiseur de requête est un plan d'exécution de requête, parfois appelé plan de requête ou plan d'exécution. Le contenu d'un plan d'exécution est détaillé plus loin dans cette rubrique.

Les entrées et les sorties de l'optimiseur de requête pendant l'optimisation d'une instruction SELECT unique sont illustrées dans le diagramme suivant :



Une instruction SELECT ne définit que :

- le format du jeu de résultats. Il est principalement spécifié dans la liste de sélection. Toutefois, d'autres clauses telles que ORDER BY et GROUP BY influencent également la syntaxe finale du jeu de résultats.
- les tables contenant les données source. Ceci est spécifié dans la clause FROM .
- la manière dont les tables sont reliées de façon logique pour les besoins de l'instruction SELECT . Elle est définie dans les spécifications de jointure, qui peuvent être présentes dans la clause WHERE ou dans une clause ON à la suite de FROM.
- Les conditions auxquelles doivent répondre les lignes des tables sources afin de correspondre à l'instruction SELECT . Elles sont spécifiées dans les clauses WHERE et HAVING .

Un plan d'exécution de requête permet de définir :

- L'ordre d'accès aux tables sources. Pour créer le jeu de résultats, le serveur de bases de données peut accéder aux tables de base selon de nombreux ordres différents. Par exemple, si l'instruction SELECT fait référence à trois tables, le serveur de base de données accédera d'abord à TableA, utilisera les données de TableA pour extraire les lignes correspondantes de TableB, puis utilisera les données de TableB pour extraire les données de TableC. Les autres séquences dans lesquelles le serveur de bases de données peut accéder aux tables sont les suivantes :

  TableC, TableB, TableAou
  TableB, TableA, TableCou
  TableB, TableC, TableAou
  TableC, TableA, TableB

- Les méthodes utilisées pour extraire les données des différentes tables.
  Il existe également différentes méthodes d'accès aux données dans chaque table. Si seules quelques lignes ayant des valeurs de clés spécifiques sont nécessaires, le serveur de base de données peut utiliser un index. Si toutes les lignes de la table sont nécessaires, le serveur de base de données peut ignorer les index et procéder à une analyse de la table. Si toutes les lignes de la table sont nécessaires mais qu'il existe un index dont les colonnes clés se trouvent dans une clause ORDER BY, l'analyse d'index plutôt que l'analyse de table peut éviter un tri séparé du jeu de résultats. Dans le cas d'une table très petite, les analyses de table peuvent s'avérer plus efficaces pour quasiment tous les accès à la table.

- Les méthodes utilisées pour effectuer les calculs, filtrer, agréger et trier les données des différentes tables.
  À mesure que les données sont consultées à partir des tables, différentes méthodes permettent d'effectuer des calculs sur les données, par exemple calculer des valeurs scalaires, et agréger et trier les données comme défini dans le texte de la requête, par

exemple en utilisant une clause GROUP BY ou ORDER BY, et filtrer les données, par exemple en utilisant une clause WHERE ou HAVING.

Le processus de sélection d'un plan d'exécution parmi plusieurs possibles est appelé optimisation. L'optimiseur de requête est un des composants les plus importants de Moteur de base de données. Bien que l'optimiseur de requête puisse créer une certaine surcharge pour analyser la requête et sélectionner un plan, celle-ci est en général largement compensée par l'adoption d'un plan d'exécution efficace. Prenons l'exemple de deux entrepreneurs en bâtiment à qui l'on commande la même maison. Si l'un d'eux commence par consacrer quelques jours à planifier la construction de cette maison alors que l'autre lance immédiatement la construction sans aucune planification, il est fort probable que celui qui a pris le temps de planifier son projet finira le premier.

L'optimiseur de requête SQL Server est un optimiseur basé sur les coûts. À chaque plan d'exécution possible est associé un coût exprimé en termes de quantité de ressources informatiques utilisées. L'optimiseur de requêtes doit analyser les plans possibles et opter pour celui dont le coût estimé est le plus faible. Certaines instructions SELECT complexes disposent de milliers de plans d'exécution possibles. Dans ce cas, l'optimiseur de requêtes n'analyse pas toutes les combinaisons possibles. Il recourt alors à des algorithmes sophistiqués afin de trouver un plan d'exécution dont le coût se rapproche raisonnablement du minimum possible.

L'optimiseur de requête SQL Server choisit non seulement le plan d'exécution dont le coût en ressources est le plus faible, mais également celui qui retourne le plus rapidement les résultats à l'utilisateur moyennant un coût en ressources raisonnable. Par exemple, le traitement d'une requête en parallèle monopolise généralement davantage de ressources qu'un traitement en série, mais il est plus rapide. L'optimiseur de requête SQL Server utilise un plan d'exécution en parallèle pour retourner les résultats si la charge du serveur n'en est pas affectée de façon rédhibitoire.

L'optimiseur de requête SQL Server se base sur les statistiques de distribution lors de l'estimation du coût en ressources des différentes méthodes d'extraction d'informations à partir d'une table ou d'un index. Les statistiques de distribution sont conservées pour les colonnes et les index, et contiennent des informations sur la densité des données sous-jacentes. Elles indiquent la sélectivité des valeurs dans un index ou une colonne spécifique. Par exemple, dans une table représentant des voitures, plusieurs voitures proviennent du même constructeur mais chacune a un numéro d'identification unique. Un index sur le numéro d'identification du véhicule est plus sélectif qu'un index sur le constructeur, car le numéro d'identification du véhicule a une plus faible densité que le constructeur. Si les statistiques d'index ne sont pas à jour, l'optimiseur de requêtes peut ne pas effectuer le meilleur choix pour l'état actuel de la table. Pour plus d'informations sur les densités, consultez Statistiques.

La densité définit la distribution des valeurs uniques qui existent dans les données ou le nombre moyen des valeurs en double pour une colonne donnée. Lorsque la densité diminue, la sélectivité d'une valeur augmente.

L'optimiseur de requête SQL Server est important, car il permet l'ajustement dynamique du serveur de base de données au fur et à mesure que la base de données évolue sans recourir à l'intervention d'un programmeur ou d'un administrateur de bases de données. Cela permet aux programmeurs de se concentrer sur la description du résultat final de la requête. Ils peuvent faire confiance à l'optimiseur de requête SQL Server dans son choix d'un plan d'exécution efficace pour l'état de la base de données à chaque exécution de l'instruction.

Sitographie :

- https://sqlpro.developpez.com/cours/quoi-indexer/
- https://fr.wikipedia.org/wiki/Microsoft_SQL_Server#Index
- https://docs.microsoft.com/fr-fr/sql/relational-databases/indexes/indexes?view=sql-server-ver15
- https://www.geeksforgeeks.org/intro-to-sql-server-architecture/
- https://www.sqlservercentral.com/blogs/sql-server-oracle-architectural-comparison
- https://blog.sqlauthority.com/2016/05/17/comparison-logical-architecture-oracle-sql-server/
- https://docs.microsoft.com/fr-fr/sql/relational-databases/memory-management-architecture-guide?view=sql-server-ver15
- https://blog.ippon.fr/2021/04/19/sql-server-2019/