4ème année – MASTER 1 MIAGE

ECUE – FONCTIONNEMENT DES SGBD

PROJET DE RECHERCHE SUR LES SGBD RELATIONNELS : MYSQL

Enseignant : Monsieur Gregory Galli

Étudiants : ***DIOP Serigne Rawane*** & ***BORREANI THEO & DIALLO Mouhamed Gando & PRUDENT Arthur & BREZZO Jérémie***

Sommaire :

Introduction :

MySQL est le plus connu des SGBD. C'est le plus utilisé, car il était (auparavant) open-source (ce qui veut dire que son code et son utilisation étaient gratuits). J'insiste sur le "auparavant", car MySQL a depuis été racheté par Oracle Corporation, et n'est plus open-source. Néanmoins, il en existe une "copie" open-source appelée MariaDB, qui suit les mêmes règles de langage que MySQL (et que je vous recommande).

MySQL est surtout connu pour être le SGBD utilisé par les sites WordPress, ce qui a fait grandement augmenter sa popularité.

Malgré cette popularité, c'est un peu la "tête de mule" des SGBD. Il ne suit pas rigoureusement la syntaxe préconisée par le SQL. Il est aussi parfois trop "permissif" sur les requêtes SQL, provoquant des erreurs.

Mais c'est un SGBD robuste, qui fonctionne très bien, même sur de grandes quantités de données ! Par exemple, c'était pendant longtemps le seul SGBD utilisé par Facebook.

1. Identification du SGBD :

Nom	MySQL
Date de parution	23/05/1995
Fondateurs	David AXMARK, Michael WIDENIUS, Allan LARSSON
Entreprise fondatrice	MySQLLAB
Entreprise propriétaire	Oracle Corporation (rachat en 2010)
Modèles de données supportés	Modèle de données relationnel
Licence	Licence Publique Générale GNU (GPL) OU Licence Commerciale

Historique des propriétaire :

- 23/05/1995 MySQLLAB
- 16/01/2008 Sun Microsystems rachète MySQLLAB donc obtiens MySQL
- 21/01/2010 Oracle Corporation rachète Sun Microsystem donc obtiens MySQL

2. Architecture fonctionnelle du SGBD choisi :

a. Architecture :

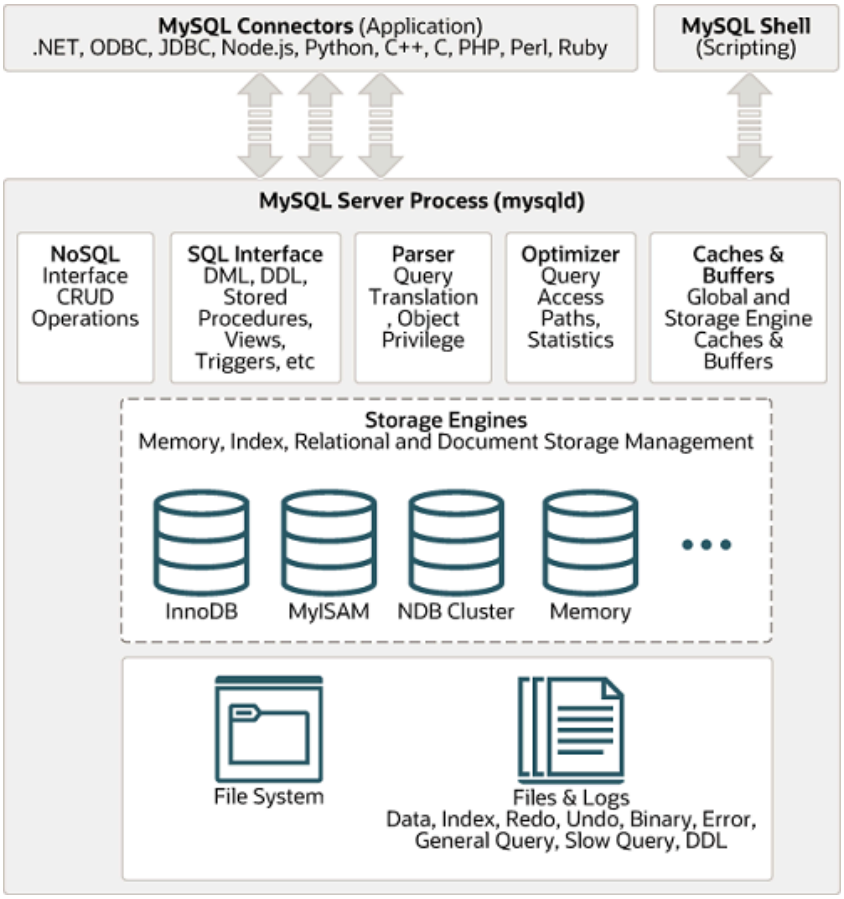
L'architecture du serveur MySQL isole le programmeur d'applications et l'administrateur de base de données de tous les détails d'implémentation de bas niveau au niveau du stockage, fournissant ainsi un modèle d'application et une API cohérents et simples. Ainsi, bien qu'il existe différentes capacités selon les différents moteurs de stockage, l'application est protégée de ces différences.

MySQL est un système de gestion de base de données relationnelle avec un type d'architecture en couches.
Il y a 3 niveaux :

- CLIENT-END, cette couche est la partie qui s'occupe de l'interaction entre les utilisateurs et la SGBD grâce à l'écran d'interface graphique ou l'invite de commande pour la soumission de différentes commandes MySQL.
- SERVER-END, cette couche correspond à la partie logique du SGBD il reçoit toutes les demandes envoyées par le client et lui envoie des réponses après le traitement de chaque demande.
- STORAGE-END, contient chaque table créée et dispose d'une API qui aide à l'exécution des requêtes.

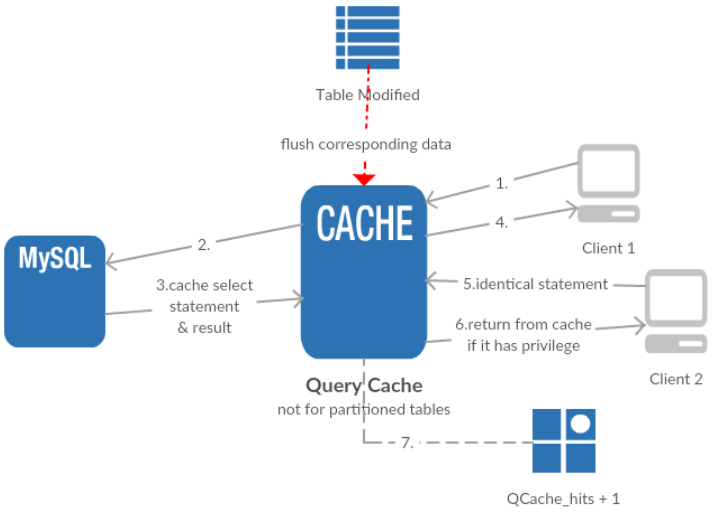
L'architecture permet au SGBD d'être efficace et modulaire offre des avantages notamment aux utilisateurs qui souhaite cibler spécifiquement un besoin d'application particulier, comme l'entretreprossage de donnée, le traitement des transactions tout en profitant de l'avantage d'utiliser un ensemble d'interfaces et de service indépendants de chacun.

Le programmeur d'application et l'administrateur de base de données interagissent avec la base de données MySQL grâce aux API du connecteur et les couches de service situées au-dessus des moteurs de stockage. Si les modifications apportées à l'application entraînent des exigences qui exigent un changement du moteur de stockage sous-jacent, ou l'ajout d'un ou plusieurs moteurs de stockage pour prendre en charge de nouveaux besoins, aucune modification significative du codage ou du processus n'est requise pour que les choses fonctionnent. L'architecture du serveur MySQL protège l'application de la complexité sous-jacente du moteur de stockage en présentant une API cohérente et facile à utiliser qui s'applique à tous les moteurs de stockage.



b. Les différents caches mémoires et leurs rôles :

L'utilité du cache mémoire est de gagner en rapidité en évitant d'aller chercher de la donnée dans la mémoire dur. Par contre en cas de panne l'entièreté de la donnée présente uniquement sur le cache est perdue.



MySQL alloue des tampons et des caches pour améliorer les performances des opérations de base de données. La configuration par défaut est modifiable pour améliorer les performances de MySQL ou les limiter.

- **Cache de requête (Query Cache)**, stocke en mémoire les résultats des requêtes déjà exécutées. MySQL Query Cache permet d'accélérer les requêtes identiques ultérieures en évitant de les exécuter à nouveau. Cela peut améliorer les performances globales du serveur de base de données en réduisant la charge et en diminuant le temps de réponse pour les requêtes fréquentes.
- **Cache des index de tables InnoDB (InnoDB Buffer Pool) ,** un cache InnoDB comporte des tables et des index. Le pool de mémoire tampon permet d'accéder directement à partir de la mémoire aux données fréquemment utilisées, ce qui accélère le traitement. D'après le site officiel MySQL sur un serveur dédié, jusqu'à 80 % de la mémoire physique est souvent affectée au pool de mémoire tampon.
- **Cache de requêtes préparées (Prepared Statement Cache) ,** un cache de requête préparé à pour but de conserver des instructions et d'ainsi exécuter plus rapidement les requête présente dans le cache.
- **Cache de clés (Key Cache),** un cache de clés est une entrée d'index qui identifie de manière unique un objet dans un cache. Par défaut, les serveurs Edge mettent en cache le contenu en fonction de l'intégralité du chemin de ressource et d'une chaîne de requête.

c. Gestion des connexions

Le client envoie une demande au serveur puis le serveur accepte ou non la demande,si le serveur accepte le client est alors connecté. Lorsque le client est connecté au serveur, le client obtient son propre fil d'exécution pour sa connexion. C'est à l'aide de ce thread que toutes les requêtes du client sont exécutées.

Néanmoins plusieurs fonctionnalités sont disponibles pour protéger et assurer la bonne fonctionnalité de notre système.

- **Connection Pooling :** permet de réutiliser des connexions au lieu d'en créer de nouvelles à chaque requête.
- **Nombre de connection maximale :** le paramètre max_connection permet de limiter le nombre de connection. le paramètre connect_timeout permet de définir une durée de connexion maximum.
- **Sécurité :** les connexions et tentatives sont enregistrées dans des journaux par souci de sécurité pour être consultable si besoin.

d. Processus d'exécution des requêtes

Entre le moment où l'utilisateur exécute une requête et celui où il obtient une réponse plusieurs étapes se sont passées, nous allons essayer de les détailler ci-dessous.

1. **Envois de la requête :** Le client envoie la requête au SGBD
2. **Analyse lexicale et syntaxique :** MySQL vérifie le lexique et la syntaxe de la requête, c'est-à-dire que les termes sont valide est dans une syntaxe correcte.
3. **Optimisation de la requête :** la requête est optimisée pour optimiser son exécution.
4. **Génération du plan d'exécution :** qui détermine comment la requête sera exécutée.
5. **Accès aux données :** accès au données en suivant le plan d'exécution.
6. **Traitement des données :** les données brut récupéré nous les traitons en fonction des filtres, agrégations, tris, etc.
7. **Envoi des résultats :** pour finir les résultats des requêtes sont envoyés du système de gestion de base de données vers le client.

3. Le dictionnaire de données du moteur choisi :

MySQL Server intègre un dictionnaire de données transactionnelles qui stocke des informations sur les objets de la base de données. Dans les versions précédentes de MySQL, les données du dictionnaire étaient stockées dans des fichiers de métadonnées, des tables non transactionnelles et des dictionnaires de données spécifiques au moteur de stockage.

Les avantages du dictionnaire de données MySQL incluent :

- Simplicité d'un schéma de dictionnaire de données centralisé qui stocke uniformément les données du dictionnaire.
- Suppression du stockage de métadonnées basé sur des fichiers.
- Stockage transactionnel et sécurisé des données du dictionnaire.
- Mise en cache uniforme et centralisée pour les objets du dictionnaire.
- Une implémentation plus simple et améliorée pour certaines INFORMATION_SCHEMAtables.
- DDL atomique.

Ces tables constituent le dictionnaire de données, qui contient des métadonnées sur les objets de base de données.

- **catalogs:** Informations catalogue.
- **character_sets:** Informations sur les jeux de caractères disponibles.
- **check_constraints:** Informations sur CHECKles contraintes définies sur les tables.
- **collations:** Informations sur les classements pour chaque jeu de caractères.
- **column_statistics:** Statistiques d'histogramme pour les valeurs de colonne.
- **column_type_elements:** Informations sur les types utilisés par les colonnes.
- **columns:** Informations sur les colonnes des tableaux.
- **dd_properties:** table qui identifie les propriétés du dictionnaire de données, telles que sa version. Le serveur l'utilise pour déterminer si le dictionnaire de données doit être mis à niveau vers une version plus récente.
- **events:** Informations sur les événements du planificateur d'événements.
- **foreign_keys, foreign_key_column_usage:** Informations sur les clés étrangères.
- **index_column_usage:** Informations sur les colonnes utilisées par les index.
- **index_partitions:** Informations sur les partitions utilisées par les index.
- **index_stats:** Utilisé pour stocker les statistiques d'index dynamique générées lors ANALYZE TABLEde l'exécution.
- **indexes:** Informations sur les index de table.
- **innodb_ddl_log:** stocke les journaux DDL pour les opérations DDL sécurisées en cas de crash.
- **parameter_type_elements:** informations sur les paramètres de procédure stockée et de fonction, ainsi que sur les valeurs de retour des fonctions stockées.
- **parameters:** Informations sur les procédures et fonctions stockées.
- **resource_groups:** Informations sur les groupes de ressources.
- **routines:** Informations sur les procédures et fonctions stockées.
- **schemata:** Informations sur les schémas. Dans MySQL, un schéma est une base de données, ce tableau fournit donc des informations sur les bases de données.
- **st_spatial_reference_systems:** Informations sur les systèmes de référence spatiale disponibles pour les données spatiales.
- **table_partition_values:** Informations sur les valeurs utilisées par les partitions de table.
- **table_partitions:** Informations sur les partitions utilisées par les tables.

- **table_stats:** Informations sur les statistiques de table dynamique générées lors ANALYZE TABLEde l'exécution.
- **tables:** Informations sur les tables des bases de données.
- **tablespace_files:** Informations sur les fichiers utilisés par les tablespaces.
- **tablespaces:** Informations sur les tablespaces actifs.
- triggers: Informations sur les déclencheurs.
- **view_routine_usage:** Informations sur les dépendances entre les vues et les fonctions stockées qu'elles utilisent.
- **view_table_usage:** Utilisé pour suivre les dépendances entre les vues et leurs tables sous-jacentes.

Les tables du dictionnaire de données sont invisibles. Ils ne peuvent pas être lus avec SELECT, n'apparaissent pas dans la sortie de SHOW TABLES, ne sont pas répertoriés dans le INFORMATION_SCHEMA.TABLES tableau, etc. Cependant, dans la plupart des cas, des INFORMATION_SCHEMA tables correspondantes peuvent être interrogées. Conceptuellement, le INFORMATION_SCHEMA fournit une vue à travers laquelle MySQL expose les métadonnées du dictionnaire de données. Par exemple, vous ne pouvez pas sélectionner mysql.schemata directement dans le tableau :

```
mysql> SELECT * FROM mysql.schemata;
ERROR 3554 (HY000): Access to data dictionary table 'mysql.schemata' is rejected.
```

Sélectionnez plutôt ces informations dans le INFORMATION_SCHEMA tableau correspondant :

```
mysql> SELECT * FROM INFORMATION_SCHEMA.SCHEMATA\G
*************************** 1. row ***************************
              CATALOG_NAME: def
               SCHEMA_NAME: mysql
DEFAULT_CHARACTER_SET_NAME: utf8mb4
    DEFAULT_COLLATION_NAME: utf8mb4_0900_ai_ci
                  SQL_PATH: NULL
        DEFAULT_ENCRYPTION: NO
*************************** 2. row ***************************
              CATALOG_NAME: def
               SCHEMA_NAME: information_schema
DEFAULT_CHARACTER_SET_NAME: utf8mb3
    DEFAULT_COLLATION_NAME: utf8mb3_general_ci
                  SQL_PATH: NULL
        DEFAULT_ENCRYPTION: NO
*************************** 3. row ***************************
              CATALOG_NAME: def
               SCHEMA_NAME: performance_schema
DEFAULT_CHARACTER_SET_NAME: utf8mb4
    DEFAULT_COLLATION_NAME: utf8mb4_0900_ai_ci
                  SQL_PATH: NULL
        DEFAULT_ENCRYPTION: NO
...
```

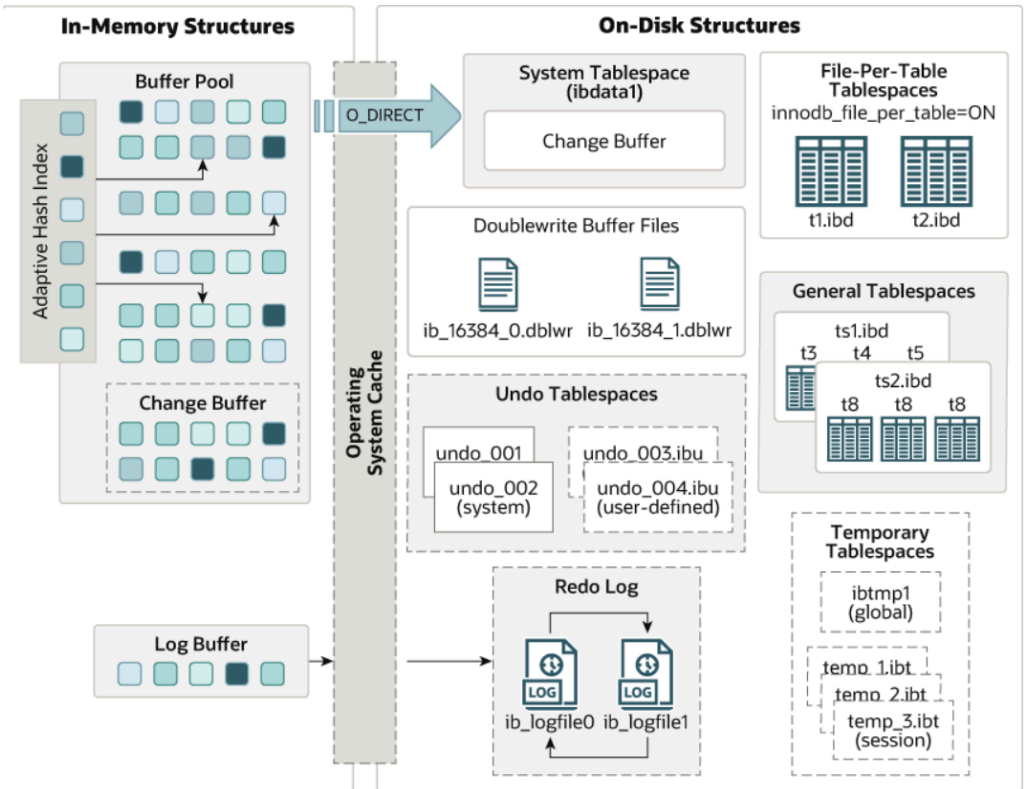
4. Organisation physique du SGBD :

InnoDB est un moteur de stockage à usage général qui équilibre haute fiabilité et hautes performances. Dans MySQL 8.0, InnoDB est le moteur de stockage MySQL par défaut.

Avantages d'InnoDB :

- Ses opérations DML suivent le modèle ACID, avec des transactions dotées de capacités de validation, de restauration et de récupération après incident pour protéger les données des utilisateurs.
- Le verrouillage au niveau des lignes et les lectures cohérentes de style Oracle augmentent la concurrence et les performances multi-utilisateurs.
- InnoDB organisent vos données sur le disque pour optimiser les requêtes basées sur les clés primaires. Chaque InnoDB table possède un index de clé primaire appelé index clusterisé qui organise les données afin de minimiser les E/S pour les recherches de clé primaire.
- Pour maintenir l'intégrité des données, InnoDB prend en charge FOREIGN KEY les contraintes. Avec les clés étrangères, les insertions, les mises à jour et les suppressions sont vérifiées pour garantir qu'elles n'entraînent pas d'incohérences entre les tables associées.

Le diagramme suivant montre les structures en mémoire et sur disque qui composent l' InnoDB architecture du moteur de stockage.



Structures en mémoire InnoDB

- **Buffer pool**
  Le pool de mémoire tampon est une zone de la mémoire principale où InnoDB les données de table et d'index sont mises en cache lors de leur accès. Le pool de mémoire tampon permet d'accéder directement à partir de la mémoire aux données fréquemment utilisées, ce qui accélère le traitement. Sur les serveurs dédiés, jusqu'à 80 % de la mémoire physique est souvent affectée au pool de mémoire tampon.

- **Change Buufer**
  Le tampon de modification est une structure de données spéciale qui met en cache les modifications apportées aux pages d'index secondaire lorsque ces pages ne se trouvent pas dans le pool de tampons . Les modifications mises en mémoire tampon, qui peuvent résulter d' opérations INSERT, UPDATEou DELETE(DML), sont fusionnées ultérieurement lorsque les pages sont chargées dans le pool de mémoire tampon par d'autres opérations de lecture.

- **Adaptive Hash Index**

  L'index de hachage adaptatif permet InnoDB de fonctionner davantage comme une base de données en mémoire sur des systèmes avec des combinaisons appropriées de charge de travail et suffisamment de mémoire pour le pool de mémoire tampon sans sacrifier les fonctionnalités transactionnelles ou la fiabilité.

- **Log Buffer**

  Le tampon de journal est la zone mémoire qui contient les données à écrire dans les fichiers journaux sur le disque. Le contenu du tampon de journal est périodiquement vidé sur le disque. Un tampon de journal volumineux permet d'exécuter des transactions volumineuses sans qu'il soit nécessaire d'écrire des données de journalisation sur le disque avant la validation des transactions. Ainsi, si vous avez des transactions qui mettent à jour, insérant ou supprimant de nombreuses lignes, l'augmentation de la taille du tampon de journal permet d'économiser les E/S disques.

Structures sur disque InnoDB

- **The System Tablespace**

  L'espace table système est la zone de stockage du tampon de modification. Il peut également contenir des données de table et d'index si les tables sont créées dans l'espace de table système plutôt que dans des espaces de table fichier par table ou généraux.

- **File-Per-Table Tablespaces**

  Il contient des données et des index pour une seule InnoDB- table et est stocké sur le système de fichiers dans un seul fichier de données.

- **General Tablespaces**

  Un tablespace général est un InnoDB tablespace partagé créé à l'aide CREATE TABLESPACEde la syntaxe.

- **Undo Tablespaces**

  Les tablespaces d'annulation contiennent des journaux d'annulation, qui sont des collections d'enregistrements contenant des informations sur la manière d'annuler la dernière modification apportée par une transaction à un enregistrement d'index clusterisé.

- **Temporary Tablespaces**

  InnoDB utilise des tablespaces temporaires de session et un tablespace temporaire global.

  - **Tablespaces temporaires de session**

    Les espaces de table temporaires de session stockent les tables temporaires créées par l'utilisateur et les tables temporaires internes créées par l'optimiseur lorsqu'il InnoDB est configuré comme moteur de stockage pour les tables temporaires internes sur disque.

  - **Espace de table temporaire global**

    L'espace de table temporaire global (ibtmp1) stocke les segments d'annulation pour les modifications apportées aux tables temporaires créées par l'utilisateur.

- **Doublewrite Buffer**

  Le tampon de double écriture est une zone de stockage dans laquelle InnoDB écrit les pages vidées du pool de mémoire tampon avant d'écrire les pages à leurs emplacements appropriés dans les InnoDB fichiers de données.

- **Redo Log**

  Le journal redo est une structure de données sur disque utilisée lors d'une récupération après incident pour corriger les données écrites par des transactions incomplètes. Pendant les opérations normales, le journal redo encode les demandes de modification des données de table résultant d'instructions SQL ou d'appels d'API de bas niveau. Les modifications qui n'ont pas terminé la mise à jour des fichiers de données avant un arrêt inattendu sont relues automatiquement lors de l'initialisation et avant l'acceptation des connexions.

- **Undo Logs**

  Un journal d'annulation est une collection d'enregistrements de journal d'annulation associés à une seule transaction de lecture-écriture. Un enregistrement de journal d'annulation contient des informations sur la manière d'annuler la dernière modification apportée par une transaction à un enregistrement d'index clusterisé . Si une autre transaction doit voir les données d'origine dans le cadre d'une opération de lecture cohérente, les données non modifiées sont récupérées à partir des enregistrements du journal d'annulation.

5. Organisation logique des données :

Le modèle logique des données consiste à décrire la structure de données utilisée sans faire référence à un langage de programmation. Il s'agit donc de préciser le type de données utilisées lors des traitements.

Ainsi, le modèle logique est dépendant du type de base de données utilisé. Ainsi dans MySql on va retrouver ce modèle logique de données.

Tables, lignes et colonnes :

Lorsque des données ont la même structure (comme par exemple, les renseignements relatifs aux clients), on peut les organiser en table dans laquelle les colonnes décrivent les champs en commun et les lignes contiennent les valeurs de ces champs pour chaque enregistrement (tableau 28).

numéro client	nom	prénom	adresse
1	Dupont	Michel	127, rue...
2	Durand	Jean	314, boulevard...
3	Dubois	Claire	51, avenue...
4	Dupuis	Marie	2, impasse...
...	...	...	...

Tableau 28 - Contenu de la table clients, avec en première ligne les intitulés des colonnes

Clés primaires et clés étrangères

Les lignes d'une table doivent être uniques, cela signifie qu'une colonne (au moins) doit servir à les identifier. Il s'agit de la clé primaire de la table.
L'absence de valeur dans une clé primaire ne doit pas être autorisée. Autrement dit, la valeur vide (NULL) est interdite dans une colonne qui sert de clé primaire, ce qui n'est pas forcément le cas des autres colonnes, dont certaines peuvent ne pas être renseignées sur toutes les lignes.

De plus, la valeur de la clé primaire d'une ligne ne devrait pas, en principe, changer au cours du temps.

Par ailleurs, il se peut qu'une colonne Colonne1 d'une table ne doit contenir que des valeurs prises par la colonne Colonne2 d'une autre table (par exemple, le numéro du client sur une commande doit correspondre à un vrai numéro de client). 2 doit être sans doublons (bien souvent il s'agit d'une clé primaire). On dit alors que 1 est une clé étrangère et qu'elle référence 2.

Par convention, on souligne les clés primaires et on fait précéder les clés étrangères d'un dièse # dans la description des colonnes d'une table :

clients(numéro client, nom client, prénom, adresse client)

commandes(numéro commande, date de commande, #numéro client (non vide))

Remarques :

- une même table peut avoir plusieurs clés étrangères mais une seule clé primaire (éventuellement composées de plusieurs colonnes) ;
- une colonne clé étrangère peut aussi être primaire (dans la même table) ;
- une clé étrangère peut être composée (c'est le cas si la clé primaire référencée est composée) ;
- implicitement, chaque colonne qui compose une clé primaire ne peut pas recevoir la valeur vide (NULL interdit) ;
- par contre, si une colonne clé étrangère ne doit pas recevoir la valeur vide, alors il faut le préciser dans la description des colonnes.

MySql vérifie au coup par coup que chaque clé étrangère ne prend pas de valeurs en dehors de celles déjà prises par la ou les colonne(s) qu'elle référence. Ce mécanisme qui agit lors de l'insertion, de la suppression ou de la mise à jour de lignes dans les tables, garantit ce que l'on appelle l'intégrité référentielle des données.

Le schéma relationnel

On peut représenter les tables d'une base de données relationnelle telle que MySql par un schéma relationnel dans lequel les tables sont appelées relations et les liens entre les clés étrangères et leur clé primaire est symbolisé par un connecteur (figure 29).

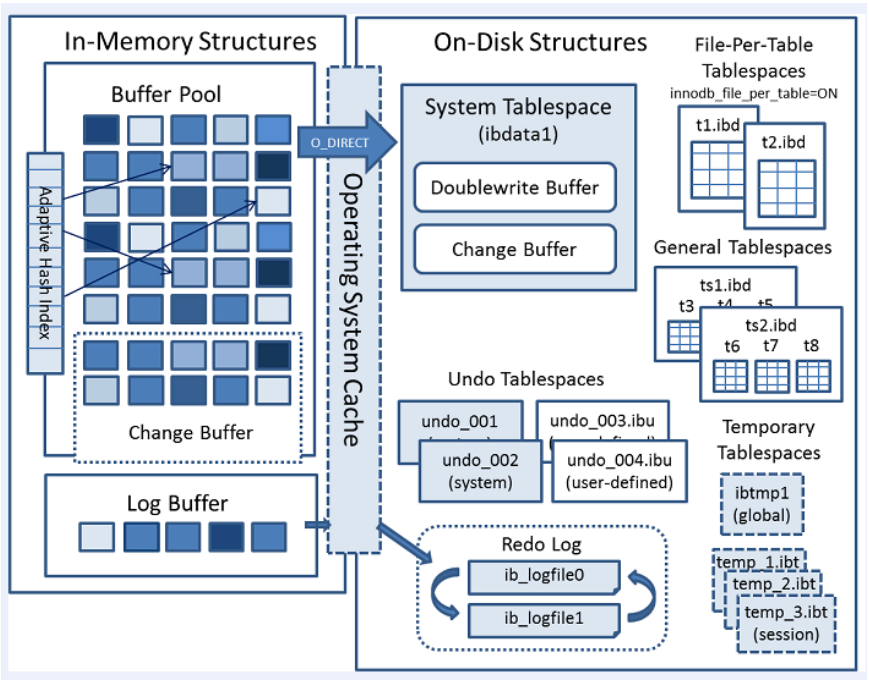
Figure 29 - Schéma relationnel simple entre deux tables

Concept de Tablespace

Un tablespace est une unité logique qui sert à regrouper des données physiques, telles que les fichiers sur disque. Il représente l'espace de stockage où les données réelles de la base de données sont conservées.

Un tablespace dans MySQL, particulièrement pour le moteur de stockage InnoDB, est un concept central pour la gestion de l'espace disque utilisé par les données de la base de données.

Dans InnoDB, le tablespace principal est généralement stocké dans un ou plusieurs fichiers sur le disque. Le fichier le plus courant est ibdata1.



Ces fichiers contiennent non seulement les données et les index des tables, mais aussi d'autres informations telles que les données d'annulation (undo logs) et les informations de gestion interne.

Pour éviter d'avoir un grand tablespace système, envisagez d'utiliser des tablespaces fichier par table ou des tablespaces généraux pour vos données. Les tablespaces fichier par table sont le type de tablespace par défaut et sont utilisés implicitement lors de la création d'une table InnoDB. Contrairement aux tablespaces système, les tablespaces fichier par table restituent de l'espace disque au système d'exploitation lorsqu'ils sont tronqués ou supprimés.

Les tablespaces généraux sont des tablespaces multi-tables qui peuvent également être utilisés comme alternative au tablespace système.

L'espace de tables du système

Le tablespace système est la zone de stockage du tampon de modification. Il peut également contenir des données de table et d'index si les tables sont créées dans le tablespace système plutôt que dans les tablespaces généraux ou de fichiers par table. Dans les versions précédentes de MySQL, le tablespace système contenait le dictionnaire de données InnoDB. Dans MySQL 8.0, InnoDB stocke les métadonnées dans le dictionnaire de données MySQL. Dans les versions précédentes de MySQL, le tablespace système contenait également la zone de stockage du tampon de double écriture. Cette zone de stockage réside dans des fichiers de double écriture séparés depuis MySQL 8.0.20.

Le tablespace système peut avoir un ou plusieurs fichiers de données. Par défaut, un seul fichier de données du tablespace système, nommé ibdata1, est créé dans le répertoire data. La taille et le nombre de fichiers de données du tablespace système sont définis par l'option de démarrage innodb_data_file_path.

Redimensionnement de l'espace de tables du système

- **Augmentation de la taille de l'espace de tables du système**

La manière la plus simple d'augmenter la taille du tablespace système est de le configurer pour qu'il s'étende automatiquement. Pour ce faire, spécifiez l'attribut autoextend pour le dernier fichier de données dans le paramètre innodb_data_file_path et redémarrez le serveur.

Par exemple :

innodb_data_file_path=ibdata1:10M:autoextend

Lorsque l'attribut autoextend est spécifié, la taille du fichier de données augmente automatiquement par incréments de 8 Mo au fur et à mesure que l'espace est requis. La variable innodb_autoextend_increment contrôle la taille de l'incrément.

Vous pouvez également augmenter la taille du tablespace système en ajoutant un autre fichier de données.

Pour ce faire, procédez comme suit

Arrêtez le serveur MySQL.

Si le dernier fichier de données dans le paramètre innodb_data_file_path est défini avec l'attribut autoextend, supprimez-le et modifiez l'attribut size pour refléter la taille actuelle du fichier de données. Pour déterminer la taille appropriée du fichier de données à spécifier, vérifiez la taille du fichier dans votre système de fichiers et arrondissez cette valeur au MB le plus proche, un MB étant égal à 1024 x 1024 octets.

Ajoutez un nouveau fichier de données au paramètre innodb_data_file_path, en spécifiant éventuellement l'attribut autoextend. L'attribut autoextend ne peut être spécifié que pour le dernier fichier de données dans le paramètre innodb_data_file_path.

Démarrez le serveur MySQL.

Par exemple, ce tablespace possède un fichier de données à extension automatique :

- **Diminution de la taille de l'espace de tables du système InnoDB**

La réduction de la taille d'un tablespace système existant n'est pas prise en charge. La seule option pour obtenir un tablespace système plus petit est de restaurer vos données à partir d'une sauvegarde vers une nouvelle instance MySQL créée avec la configuration de taille de tablespace système souhaitée.

Pour éviter d'avoir un grand tablespace système, envisagez d'utiliser des tablespaces fichier par table ou des tablespaces généraux pour vos données. Les tablespaces fichier par table sont le type de tablespace par défaut et sont utilisés implicitement lors de la création d'une table InnoDB. Contrairement aux tablespaces système, les tablespaces fichier par table restituent de l'espace disque au système d'exploitation lorsqu'ils sont tronqués ou supprimés. Les tablespaces généraux sont des tablespaces multi-tables qui peuvent également être utilisés comme alternative au tablespace système.

File-Per-Table Tablespaces

Un tablespace fichier par table contient des données et des index pour une seule table InnoDB et est stocké sur le système de fichiers dans un seul fichier de données.

InnoDB crée par défaut des tables dans des tablespaces fichier par table. Ce comportement est contrôlé par la variable innodb_file_per_table. En désactivant la variable innodb_file_per_table, InnoDB crée des tables dans le tablespace système.

Le paramètre innodb_file_per_table peut être spécifié dans un fichier d'options ou configuré à l'exécution à l'aide d'une instruction SET GLOBAL. La modification du paramètre au moment de l'exécution nécessite des privilèges suffisants pour définir des variables système globales.

[mysqld]

innodb_file_per_table=ON

Utilisation de SET GLOBAL au moment de l'exécution

mysql> SET GLOBAL innodb_file_per_table=ON;

Un tablespace fichier par table est créé dans un fichier de données .ibd dans un répertoire de schéma sous le répertoire de données MySQL. Le fichier .ibd porte le nom de la table (nom_de_la_table.ibd). Par exemple, le fichier de données pour la table test.t1 est créé dans le répertoire test sous le répertoire de données MySQL :

mysql> USE test;

mysql> CREATE TABLE t1 (

  id INT PRIMARY KEY AUTO_INCREMENT,

  name VARCHAR(100)

) ENGINE = InnoDB;

$> cd /*path*/*to*/*mysql*/data/test

$> ls

t1.ibd

Espaces de table généraux

Un tablespace général est un tablespace InnoDB partagé qui est créé à l'aide de la syntaxe CREATE TABLESPACE.

- **Capacités générales du Tablespace**

Les tablespaces généraux sont créés à l'aide de la syntaxe CREATE TABLESPACE.

CREATE TABLESPACE *tablespace_name*

[ADD DATAFILE '*file_name*']

[FILE_BLOCK_SIZE = *value*]

[ENGINE [=] *engine_name*]

Un tablespace général peut être créé dans le répertoire de données ou en dehors de celui-ci. Pour éviter les conflits avec les tablespaces de fichiers par table créés

implicitement, la création d'un tablespace général dans un sous-répertoire du répertoire data n'est pas prise en charge. Lors de la création d'un tablespace général en dehors du répertoire de données, le répertoire doit exister et être connu d'InnoDB avant la création du tablespace. Pour qu'un répertoire inconnu soit connu d'InnoDB, ajoutez-le à la valeur de l'argument innodb_directories. innodb_directories est une option de démarrage en lecture seule. Pour la configurer, il faut redémarrer le serveur.

La clause ADD DATAFILE est optionnelle depuis MySQL 8.0.14 et obligatoire auparavant. Si la clause ADD DATAFILE n'est pas spécifiée lors de la création d'un tablespace, un fichier de données de tablespace avec un nom de fichier unique est créé implicitement. Le nom de fichier unique est un UUID de 128 bits formaté en cinq groupes de nombres hexadécimaux séparés par des tirets (aaaaaaaa-bbbb-cc-dddd-eeeeeeeeeeee). Les fichiers de données du tablespace général portent l'extension .ibd. Dans un environnement de réplication, le nom du fichier de données créé sur la source n'est pas le même que le nom du fichier de données créé sur le réplica.

- **Création d'un Tablespace général**

Les tablespaces généraux prennent en charge tous les formats de lignes de table (REDUNDANT, COMPACT, DYNAMIC, COMPRESSED), à ceci près que les tables compressées et non compressées ne peuvent pas coexister dans le même tablespace général en raison de la différence de taille des pages physiques.

Pour qu'un tablespace général contienne des tables compressées (ROW_FORMAT=COMPRESSED), l'option FILE_BLOCK_SIZE doit être spécifiée, et la valeur FILE_BLOCK_SIZE doit être une taille de page compressée valide par rapport à la valeur innodb_page_size. De plus, la taille de page physique de la table compressée (KEY_BLOCK_SIZE) doit être égale à FILE_BLOCK_SIZE/1024. Par exemple, si innodb_page_size=16KB et FILE_BLOCK_SIZE=8K, la KEY_BLOCK_SIZE de la table doit être de 8.

Le tableau suivant présente les combinaisons autorisées de innodb_page_size, FILE_BLOCK_SIZE et KEY_BLOCK_SIZE. Les valeurs de FILE_BLOCK_SIZE peuvent également être spécifiées en octets. Pour déterminer une valeur KEY_BLOCK_SIZE valide pour une valeur FILE_BLOCK_SIZE donnée, divisez la valeur FILE_BLOCK_SIZE par 1024.

Table 15.3 Permitted Page Size, FILE_BLOCK_SIZE, and KEY_BLOCK_SIZE Combinations for Compressed Tables

InnoDB Page Size (innodb_page_size)	Permitted FILE_BLOCK_SIZE Value	Permitted KEY_BLOCK_SIZE Value
64KB	64K (65536)	Compression is not supported
32KB	32K (32768)	Compression is not supported
16KB	16K (16384)	None. If `innodb_page_size` is equal to `FILE_BLOCK_SIZE`, the tablespace cannot contain a compressed table.
16KB	8K (8192)	8
16KB	4K (4096)	4
16KB	2K (2048)	2
16KB	1K (1024)	1
8KB	8K (8192)	None. If `innodb_page_size` is equal to `FILE_BLOCK_SIZE`, the tablespace cannot contain a compressed table.
8KB	4K (4096)	4
8KB	2K (2048)	2
8KB	1K (1024)	1
4KB	4K (4096)	None. If `innodb_page_size` is equal to `FILE_BLOCK_SIZE`, the tablespace cannot contain a compressed table.
4KB	2K (2048)	2
4KB	1K (1024)	1

- **Ajout de tables à un Tablespace général**

Après avoir créé un tablespace général, les instructions CREATE TABLE tbl_name ... TABLESPACE [=] tablespace_name ou ALTER TABLE tbl_name TABLESPACE [=] tablespace_name peuvent être utilisées pour ajouter des tables au tablespace, comme le montrent les exemples suivants :

CREATE TABLE:

mysql> CREATE TABLE t1 (c1 INT PRIMARY KEY) TABLESPACE ts1;

ALTER TABLE:

mysql> ALTER TABLE t2 TABLESPACE ts1;

- **Déplacement de tables entre les tablespaces à l'aide de ALTER TABLE**

ALTER TABLE avec l'option TABLESPACE peut être utilisé pour déplacer une table vers un tablespace général existant, vers un nouveau tablespace fichier par table ou vers le tablespace système. Pour déplacer une table d'un tablespace fichier par table ou du tablespace système vers un tablespace général, indiquez le nom du tablespace général. Le tablespace général doit exister.

ALTER TABLE tbl_name TABLESPACE [=] *tablespace_name*;

Pour déplacer une table d'un tablespace général ou d'un tablespace fichier par table vers le tablespace système, indiquez innodb_system comme nom de tablespace.

ALTER TABLE tbl_name TABLESPACE [=] innodb_file_per_table;

ALTER TABLE ... Les opérations TABLESPACE entraînent une reconstruction complète de la table, même si l'attribut TABLESPACE n'a pas changé par rapport à sa valeur précédente.

ALTER TABLE ... La syntaxe TABLESPACE ne permet pas de déplacer une table d'un tablespace temporaire vers un tablespace persistant.

La clause DATA DIRECTORY est autorisée avec CREATE TABLE ... TABLESPACE=innodb_file_per_table mais n'est pas supportée en combinaison avec l'option TABLESPACE. Depuis MySQL 8.0.21, le répertoire spécifié dans une clause DATA DIRECTORY doit être connu d'InnoDB.

Tablespace d’annulation

Les tablespaces d'annulation contiennent des journaux d'annulation, qui sont des collections d'enregistrements contenant des informations sur la manière d'annuler la dernière modification apportée par une transaction à un enregistrement d'index en grappe.

Deux tablespaces d'annulation par défaut sont créés lors de l'initialisation de l'instance MySQL. Les tablespaces d'annulation par défaut sont créés au moment de l'initialisation afin de fournir un emplacement pour les segments de retour en arrière qui doivent exister avant que les instructions SQL puissent être acceptées. Un minimum de deux tablespaces d'annulation est nécessaire pour prendre en charge la troncature automatique des tablespaces d'annulation. Voir Troncature des tablespaces d'annulation.

Les tablespaces d'annulation par défaut sont créés dans l'emplacement défini par la variable innodb_undo_directory. Si la variable innodb_undo_directory est indéfinie, les tablespaces d'annulation par défaut sont créés dans le répertoire de données. Les fichiers de données des tablespaces d'annulation par défaut sont nommés undo_001 et undo_002. Les noms des tablespaces d'annulation correspondants définis dans le dictionnaire de données sont innodb_undo_001 et innodb_undo_002.

Annulation de la taille de l'espace de stockage

Avant MySQL 8.0.23, la taille initiale d'un tablespace d'annulation dépend de la valeur innodb_page_size. Pour la taille de page par défaut de 16 Ko, la taille initiale du fichier de l'espace de tables d'annulation est de 10 Mo. Pour des tailles de page de 4KB, 8KB, 32KB et 64KB, les tailles initiales des fichiers du tablespace d'annulation sont respectivement de 7MiB, 8MiB, 20MiB et 40MiB. Depuis MySQL 8.0.23, la taille initiale du tablespace d'annulation est normalement de 16 Mo. La

taille initiale peut être différente lorsqu'un nouvel espace de stockage d'annulation est créé par une opération de troncature. Dans ce cas, si la taille de l'extension de fichier est supérieure à 16 Mo, et que l'extension de fichier précédente a eu lieu au cours de la dernière seconde, le nouvel espace de stockage d'annulation est créé à un quart de la taille définie par la variable innodb_max_undo_log_size.

Avant MySQL 8.0.23, un tablespace d'annulation est étendu de quatre extensions à la fois. À partir de MySQL 8.0.23, un tablespace d'annulation est étendu d'un minimum de 16 Mo. Pour gérer une croissance agressive, la taille de l'extension du fichier est doublée si l'extension précédente s'est produite moins de 0,1 seconde auparavant. Le doublement de la taille de l'extension peut se produire plusieurs fois jusqu'à un maximum de 256 Mo. Si l'extension précédente a eu lieu plus de 0,1 seconde auparavant, la taille de l'extension est réduite de moitié, ce qui peut également se produire plusieurs fois, jusqu'à un minimum de 16 Mo. Si l'option AUTOEXTEND_SIZE est définie pour un tablespace d'annulation, celui-ci est étendu en fonction de la valeur la plus élevée entre le paramètre AUTOEXTEND_SIZE et la taille d'extension déterminée par la logique décrite ci-dessus.

- **Ajout de tablespaces d'annulation**

Comme les journaux d'annulation peuvent devenir volumineux lors de transactions de longue durée, la création de tablespaces d'annulation supplémentaires peut aider à éviter que les tablespaces d'annulation individuels ne deviennent trop volumineux. Depuis MySQL 8.0.14, des tablespaces d'annulation supplémentaires peuvent être créés au moment de l'exécution en utilisant la syntaxe CREATE UNDO TABLESPACE.

CREATE UNDO TABLESPACE *tablespace_name* ADD DATAFILE '*file_name*.ibu';

Le nom du fichier du tablespace d'annulation doit avoir une extension .ibu. Il n'est pas permis de spécifier un chemin relatif lors de la définition du nom de fichier du tablespace d'annulation. Un chemin d'accès entièrement qualifié est autorisé, mais il doit être connu d'InnoDB. Les chemins connus sont ceux définis par la variable innodb_directories. Il est recommandé d'utiliser des noms de fichier uniques pour l'undo tablespace afin d'éviter tout conflit de nom de fichier lors du déplacement ou du clonage de données.

Une instance MySQL supporte jusqu'à 127 tablespaces d'annulation, y compris les deux tablespaces d'annulation par défaut créés lors de l'initialisation de l'instance MySQL.

- **Suppression des tablespaces d'annulation**

Depuis MySQL 8.0.14, les tablespaces d'annulation créés à l'aide de la syntaxe CREATE UNDO TABLESPACE peuvent être supprimés à l'exécution à l'aide de la syntaxe DROP UNDO TABALESPACE.

Un tablespace d'annulation doit être vide avant de pouvoir être supprimé. Pour vider un tablespace undo, celui-ci doit d'abord être marqué comme inactif à l'aide de la syntaxe ALTER UNDO TABLESPACE, de sorte que le tablespace ne soit plus utilisé pour attribuer des segments de rollback aux nouvelles transactions.

ALTER UNDO TABLESPACE *tablespace_name* SET INACTIVE;

Une fois qu'un tablespace d'annulation est marqué comme inactif, les transactions qui utilisent actuellement des segments de rollback dans le tablespace d'annulation sont autorisées à se terminer, de même que toutes les transactions commencées avant que ces transactions ne soient terminées. Une fois les transactions terminées, le système de purge libère les segments de rollback dans le tablespace d'annulation et ce dernier est tronqué à sa taille initiale. (Le même processus est utilisé pour tronquer les tablespaces d'annulation). Une fois que l'undo tablespace est vide, il peut être supprimé.

DROP UNDO TABLESPACE *tablespace_name*;

L'état d'un tablespace d'annulation peut être contrôlé en interrogeant la table INNODB_TABLESPACES du schéma d'information.

SELECT NAME, STATE FROM INFORMATION_SCHEMA.INNODB_TABLESPACES

WHERE NAME LIKE '*tablespace_name*';

Un état inactif indique que les segments de rollback d'un tablespace d'annulation ne sont plus utilisés par les nouvelles transactions. Un état vide indique qu'un tablespace d'annulation est vide et prêt à être supprimé, ou prêt à être rendu actif à l'aide d'une instruction ALTER UNDO TABLESPACE nom_du_tableau SET ACTIVE. Toute tentative de suppression d'un tablespace d'annulation qui n'est pas vide entraîne une erreur.

Les tablespaces d'annulation par défaut (innodb_undo_001 et innodb_undo_002) créés lors de l'initialisation de l'instance MySQL ne peuvent pas être supprimés. Ils peuvent cependant être rendus inactifs en utilisant une instruction ALTER UNDO TABLESPACE tablespace_name SET INACTIVE. Avant qu'un tablespace d'annulation par défaut puisse être rendu inactif, il doit y avoir un tablespace d'annulation pour le remplacer. Un minimum de deux tablespaces d'annulation actifs est nécessaire à tout moment pour permettre la troncature automatique des tablespaces d'annulation.

Espaces de tables temporaires

InnoDB utilise des tablespaces temporaires de session et un tablespace temporaire global.

- **Espaces de tables temporaires de la session**

Les tablespaces temporaires de session stockent les tables temporaires créées par l'utilisateur et les tables temporaires internes créées par l'optimiseur lorsque InnoDB est configuré comme moteur de stockage pour les tables temporaires internes sur disque. À partir de MySQL 8.0.16, le moteur de stockage utilisé pour les tables temporaires internes sur disque est InnoDB. (Auparavant, le moteur de stockage était déterminé par la valeur de internal_tmp_disk_storage_engine).

Les tablespaces temporaires de session sont alloués à une session à partir d'un pool de tablespaces temporaires lors de la première demande de création d'une table temporaire sur disque. Un maximum de deux tablespaces est alloué à une session, l'un pour les tables temporaires créées par l'utilisateur et l'autre pour les tables temporaires internes créées par l'optimiseur. Les tablespaces temporaires alloués à une session sont utilisés pour toutes les tables temporaires sur disque créées par la session. Lorsqu'une session se déconnecte, ses tablespaces temporaires sont tronqués et remis dans le pool. Un pool de 10 tablespaces temporaires est créé au démarrage du serveur. La taille du pool ne diminue jamais et des tablespaces sont ajoutés automatiquement au pool si nécessaire. Le pool de tablespaces temporaires est supprimé lors d'un arrêt normal ou d'une initialisation interrompue. Les fichiers des tablespaces temporaires de session ont une taille de cinq pages lorsqu'ils sont créés et ont une extension de nom de fichier .ibt.

Une plage de 400 000 ID d'espace est réservée aux tablespaces temporaires de session. Comme le pool de tablespaces temporaires de session est recréé à chaque démarrage du serveur, les ID d'espace pour les tablespaces temporaires de session ne sont pas conservés lors de l'arrêt du serveur et peuvent être réutilisés.

La variable innodb_temp_tablespaces_dir définit l'emplacement où les tablespaces temporaires de session sont créées. L'emplacement par défaut est le répertoire #innodb_temp dans le répertoire de données. Le démarrage est refusé si le pool de tablespaces temporaires ne peut pas être créé.

$> cd *BASEDIR*/data/#innodb_temp

$> ls

temp_10.ibt temp_2.ibt temp_4.ibt temp_6.ibt temp_8.ibt

temp_1.ibt temp_3.ibt temp_5.ibt temp_7.ibt temp_9.ibt

En mode de réplication basée sur des instructions (SBR), les tables temporaires créées sur un réplica résident dans un tablespace temporaire à session unique qui n'est tronqué qu'à l'arrêt du serveur MySQL.

La table INNODB_SESSION_TEMP_TABLESPACES fournit des métadonnées sur les tablespaces temporaires de session. La table Information Schema INNODB_TEMP_TABLE_INFO fournit des métadonnées sur les tables temporaires créées par l'utilisateur et actives dans une instance InnoDB.

- **Espace de tables temporaire global**

Le tablespace temporaire global (ibtmp1) stocke les segments de retour en arrière pour les modifications apportées aux tables temporaires créées par l'utilisateur.

La variable innodb_temp_data_file_path définit le chemin relatif, le nom, la taille et les attributs des fichiers de données du tablespace temporaire global. Si aucune valeur n'est spécifiée pour innodb_temp_data_file_path, le comportement par défaut est de créer un seul fichier de données à extension automatique nommé ibtmp1 dans le répertoire innodb_data_home_dir. La taille initiale du fichier est légèrement supérieure à 12 Mo.

Le tablespace temporaire global est supprimé lors d'un arrêt normal ou d'une initialisation interrompue, et recréé à chaque démarrage du serveur. Le tablespace temporaire global reçoit un ID d'espace généré dynamiquement lors de sa création. Le démarrage est refusé si le tablespace temporaire global ne peut pas être créé. Le tablespace temporaire global n'est pas supprimé en cas d'arrêt inattendu du serveur. Dans ce cas, l'administrateur de la base de données peut supprimer le tablespace temporaire global manuellement ou redémarrer le serveur MySQL. Le redémarrage du serveur MySQL supprime et recrée automatiquement le tablespace temporaire global.

Le tablespace temporaire global ne peut pas résider sur un périphérique brut.

La table FILES du schéma d'information fournit des métadonnées sur le tablespace temporaire global. Lancez une requête similaire à celle-ci pour afficher les métadonnées du tablespace temporaire global :

mysql> SELECT * FROM INFORMATION_SCHEMA.FILES WHERE TABLESPACE_NAME='innodb_temporary'\G

Par défaut, le fichier de données du tablespace temporaire global s'étend automatiquement et sa taille augmente en fonction des besoins.

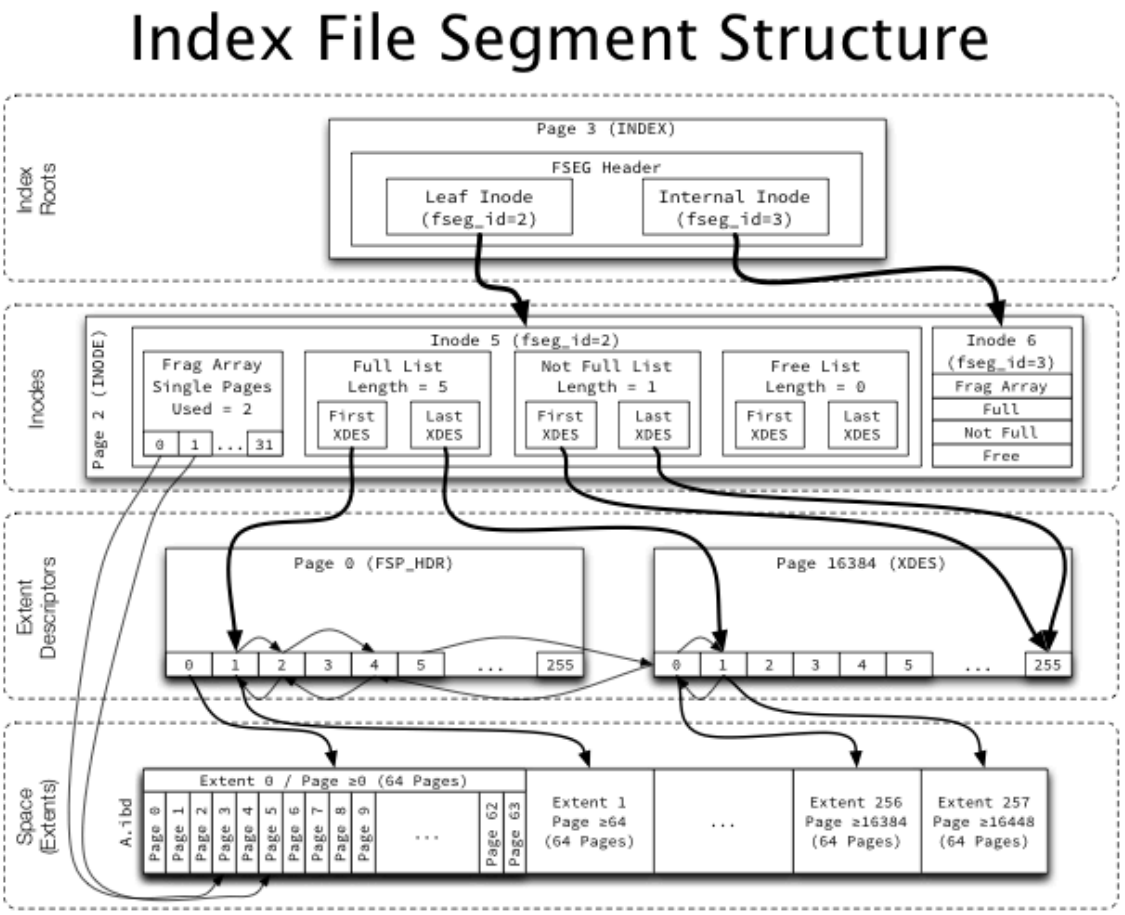
Pour déterminer si un fichier de données de l'espace de stockage temporaire global s'étend automatiquement, vérifiez le paramètre innodb_temp_data_file_path :

mysql> SELECT ***@@innodb_temp_data_file_path***;

```
+----------------------------+
| @@innodb_temp_data_file_path |
+----------------------------+
| ibtmp1:12M:autoextend      |
+----------------------------+
```

Concept de Segment

Dans MySQL, particulièrement avec le moteur de stockage InnoDB, les segments sont des composants clés de l'organisation interne des tablespaces. Ils jouent un rôle important dans la manière dont les données et les index sont stockés et gérés.



Un segment dans InnoDB est une collection d'extensions. Il représente un niveau intermédiaire dans la hiérarchie de stockage entre le tablespace et les extensions.
Chaque segment est dédié à un type spécifique de données, comme les données de la table, les index, ou les données d'annulation.
Les segments organisent le stockage au sein du tablespace, permettant une allocation et une gestion efficaces de l'espace.

- **Segment de Données** :

Ce type de segment est utilisé pour stocker les données réelles des tables.
Chaque table a son propre segment de données, surtout si la configuration 'fichier par table' est utilisée.

- **Segment d'Index** :

Les index, y compris l'index clusterisé et les index secondaires, ont également leurs propres segments. Les segments d'index permettent une recherche rapide et efficace des données.

- **Segment d'Annulation (Undo Segment)** :

Ces segments stockent les informations nécessaires pour annuler les transactions ou pour fournir des versions de données cohérentes pour les transactions en cours.

- **Segment de Rollback** :

Les segments de rollback contiennent des informations sur les opérations de rollback nécessaires en cas d'annulation de transaction.

- **Allocation d'Espace** :

L'espace dans un segment est alloué en extensions. Une extension est un groupe de pages (ou blocs) contiguës dans le tablespace.

Cette allocation permet une utilisation efficace de l'espace disque et réduit la fragmentation.

- **Extension et Rétrécissement** :

Les segments peuvent s'étendre dynamiquement lorsqu'il y a besoin de plus d'espace pour stocker des données ou des index supplémentaires.

InnoDB gère également le rétrécissement des segments, bien que cela soit moins fréquent et dépend de divers facteurs.

- **Performance et Optimisation** :

La gestion des segments influence directement les performances de la base de données, car elle affecte la manière dont les données sont lues et écrites sur le disque.

Une bonne gestion des segments peut réduire la fragmentation et optimiser les opérations d’E/S.

Les segments dans MySQL InnoDB sont essentiels pour une gestion efficace de l'espace disque et des performances. Ils permettent une organisation logique des données et des index au sein des tablespaces, facilitant ainsi l'accès aux données, leur gestion et leur optimisation. La compréhension de ces composants internes est importante pour les administrateurs de bases de données lorsqu'ils conçoivent et optimisent des systèmes de bases de données.

6. Gestion de la concurrence d'accès :

MySQL utilise principalement deux modèles pour gérer la concurrence d'accès : le modèle de verrouillage et le modèle d'isolation des transactions. Ces modèles sont conformes aux propriétés ACID (Atomicité, Cohérence, Isolation, Durabilité) des transactions dans les bases de données relationnelles.

Cette section aborde plusieurs sujets liés au verrouillage InnoDB et au modèle de transaction InnoDB..

La Section 6.1, "Verrouillage InnoDB" décrit les types de verrou utilisés par InnoDB.

Section 6.2, " Modèle de transaction InnoDB " décrit les niveaux d'isolation des transactions et les stratégies de verrouillage utilisées par chacun d'entre eux. Elle traite également de l'utilisation de l'autocommit, des lectures cohérentes non verrouillées et des lectures verrouillées.

Verrouillage InnoDB

- **Verrous partagés et exclusifs**

InnoDB met en œuvre un verrouillage standard au niveau des lignes, avec deux types de verrous : les verrous partagés (S) et les verrous exclusifs (X).

Un verrou partagé (S) permet à la transaction qui détient le verrou de lire une ligne.

Un verrou exclusif (X) permet à la transaction qui détient le verrou de mettre à jour ou de supprimer une ligne.

Si la transaction T1 détient un verrou partagé (S) sur la ligne r, les demandes d'une transaction distincte T2 pour un verrou sur la ligne r sont traitées comme suit :

Une demande de T2 pour un verrou S peut être accordée immédiatement. Par conséquent, T1 et T2 détiennent tous deux un verrou S sur la ligne R.

Une demande de T2 pour un verrou X ne peut pas être accordée immédiatement.

Si une transaction T1 détient un verrou exclusif (X) sur la ligne r, une demande d'une transaction T2 distincte pour un verrou de l'un ou l'autre type sur r ne peut pas être accordée immédiatement. Au lieu de cela, la transaction T2 doit attendre que la transaction T1 libère son verrou sur la ligne r.

	X	IX	S	IS
X	Conflict	Conflict	Conflict	Conflict
IX	Conflict	Compatible	Conflict	Compatible
S	Conflict	Conflict	Compatible	Compatible
IS	Conflict	Compatible	Compatible	Compatible

Un verrou est accordé à une transaction requérante s'il est compatible avec les verrous existants, mais pas s'il entre en conflit avec les verrous existants. Une transaction attend que le verrou existant en conflit soit libéré. Si une demande de verrou entre en conflit avec un verrou existant et ne peut être accordée parce qu'elle entraînerait un blocage, une erreur se produit.

Les verrous d'intention ne bloquent rien, à l'exception des demandes de tables complètes (par exemple, LOCK TABLES ... WRITE). L'objectif principal des verrous d'intention est de montrer que quelqu'un verrouille une ligne ou va verrouiller une ligne dans la table.

- **Verrous d'enregistrement**

Un verrou d'enregistrement est un verrou sur un enregistrement d'index. Par exemple, SELECT c1 FROM t WHERE c1 = 10 FOR UPDATE ; empêche toute autre transaction d'insérer, de mettre à jour ou de supprimer des lignes pour lesquelles la valeur de t.c1 est 10.

Les verrous d'enregistrement verrouillent toujours les enregistrements d'index, même si une table est définie sans index. Dans ce cas, InnoDB crée un index en grappe caché et l'utilise pour le verrouillage des enregistrements. Voir la Section 15.6.2.1, " Index en grappe et secondaires ".

- **Verrous d'espacement**

Un verrou d'espacement est un verrou sur un espace entre des enregistrements d'index, ou un verrou sur l'espace avant le premier ou après le dernier enregistrement d'index. Par exemple, SELECT c1 FROM t WHERE c1 BETWEEN 10 and 20 FOR UPDATE ; empêche d'autres transactions d'insérer une valeur de 15 dans la colonne t.c1, qu'il y ait ou non déjà une telle valeur dans la colonne, parce que les espaces entre toutes les valeurs existantes dans la plage sont verrouillés.

Un écart peut couvrir une seule valeur d'index, plusieurs valeurs d'index ou même être vide.

Les verrous d'espacement font partie du compromis entre les performances et la concurrence, et sont utilisés dans certains niveaux d'isolation des transactions et pas dans d'autres.

Le verrouillage de l'espace n'est pas nécessaire pour les instructions qui verrouillent les lignes en utilisant un index unique pour rechercher une ligne unique. (Cela n'inclut pas le cas où la condition de recherche n'inclut que certaines colonnes d'un index unique à plusieurs colonnes ; dans ce cas, le verrouillage de l'espace se produit).

Il convient également de noter que des verrous conflictuels peuvent être détenus sur un espace par différentes transactions. Par exemple, la transaction A peut détenir un

verrou partagé (verrou S) sur un espace alors que la transaction B détient un verrou exclusif (verrou X) sur le même espace. La raison pour laquelle des verrous d'espace conflictuels sont autorisés est que si un enregistrement est purgé d'un index, les verrous d'espace détenus sur l'enregistrement par différentes transactions doivent être fusionnés.

Les verrous d'espace dans InnoDB sont "purement inhibitifs", ce qui signifie que leur seul but est d'empêcher d'autres transactions d'insérer dans l'espace. Les verrous d'espacement peuvent coexister. Un verrou d'espace pris par une transaction n'empêche pas une autre transaction de prendre un verrou d'espace sur le même espace. Il n'y a pas de différence entre les verrous d'espace partagés et exclusifs. Ils n'entrent pas en conflit l'un avec l'autre et remplissent la même fonction.

Le verrouillage de l'espace peut être désactivé explicitement. C'est le cas si vous modifiez le niveau d'isolation de la transaction en READ COMMITTED. Dans ce cas, le verrouillage de l'espace est désactivé pour les recherches et les balayages d'index et n'est utilisé que pour le contrôle des contraintes de clés étrangères et le contrôle des clés dupliquées.

L'utilisation du niveau d'isolation READ COMMITTED a également d'autres effets. Les verrous d'enregistrement pour les lignes non correspondantes sont libérés une fois que MySQL a évalué la condition WHERE. Pour les instructions UPDATE, InnoDB effectue une lecture "semi-consistante", c'est-à-dire qu'il renvoie la dernière version validée à MySQL afin que ce dernier puisse déterminer si la ligne correspond à la condition WHERE de l'UPDATE.

- **Verrous de la clé suivante**

Un verrou de clé suivante est une combinaison d'un verrou d'enregistrement sur l'enregistrement d'index et d'un verrou d'espace sur l'espace précédant l'enregistrement d'index.

InnoDB effectue le verrouillage au niveau des lignes de telle sorte que lorsqu'il recherche ou analyse un index de table, il définit des verrous partagés ou exclusifs sur les enregistrements d'index qu'il rencontre. Ainsi, les verrous de niveau ligne sont en fait des verrous d'enregistrement d'index. Un verrou de clé suivante sur un enregistrement d'index affecte également l'"espace" avant cet enregistrement d'index. En d'autres termes, un verrou de clé suivante est un verrou d'enregistrement d'index plus un verrou d'espace sur l'espace précédant l'enregistrement d'index. Si une session dispose d'un verrou partagé ou exclusif sur l'enregistrement R d'un index, une autre session ne peut pas insérer un nouvel enregistrement d'index dans l'espace précédant immédiatement R dans l'ordre de l'index.

- **Verrous d'intention d'insertion**

Un verrou d'intention d'insertion est un type de verrou d'espace défini par les opérations INSERT avant l'insertion d'une ligne. Ce verrou signale l'intention d'insertion de telle sorte que plusieurs transactions insérant dans le même espace d'indexation n'ont pas besoin d'attendre l'une l'autre si elles n'insèrent pas à la même position dans l'espace. Supposons qu'il existe des enregistrements d'index avec des valeurs de 4 et 7. Des transactions séparées qui tentent d'insérer les valeurs 5 et 6, respectivement, verrouillent chacune l'espace entre 4 et 7 avec des verrous d'intention d'insertion avant d'obtenir le verrou exclusif sur la ligne insérée, mais ne se bloquent pas l'une l'autre parce que les lignes ne sont pas contradictoires.

L'exemple suivant illustre une transaction prenant un verrou d'intention d'insertion avant d'obtenir un verrou exclusif sur l'enregistrement inséré. L'exemple implique deux clients, A et B.

Le client A crée une table contenant deux enregistrements d'index (90 et 102), puis lance une transaction qui place un verrou exclusif sur les enregistrements d'index dont l'ID est supérieur à 100. Le verrou exclusif comprend un verrou d'espacement avant l'enregistrement 102 :

mysql> CREATE TABLE child (id int(11) NOT NULL, PRIMARY KEY(id)) ENGINE=InnoDB;

mysql> INSERT INTO child (id) values (90),(102);

mysql> START TRANSACTION;

mysql> SELECT * FROM child WHERE id > 100 FOR UPDATE;

+-----+

| id |

+-----+

| 102 |

+-----+

Le client B commence une transaction pour insérer un enregistrement dans l'espace. La transaction prend un verrou d'intention d'insertion en attendant d'obtenir un verrou exclusif.

mysql> START TRANSACTION;

mysql> INSERT INTO child (id) VALUES (101);

Les données de transaction pour un verrou d'intention d'insertion ressemblent à ce qui suit dans SHOW ENGINE INNODB STATUS et dans la sortie du moniteur InnoDB :

RECORD LOCKS space id 31 page no 3 n bits 72 index `PRIMARY` of table `test`.`child`

trx id 8731 lock_mode X locks gap before rec insert intention waiting

Record lock, heap no 3 PHYSICAL RECORD: n_fields 3; compact format; info bits 0

0: len 4; hex 80000066; asc f;;

1: len 6; hex 000000002215; asc " ;;

2: len 7; hex 9000000172011c; asc r ;;...

- **Verrous AUTO-INC**

Un verrou AUTO-INC est un verrou spécial au niveau de la table pris par les transactions qui insèrent des valeurs dans des tables avec des colonnes AUTO_INCREMENT. Dans le cas le plus simple, si une transaction insère des valeurs dans la table, toutes les autres transactions doivent attendre pour effectuer leurs propres insertions dans cette table, de sorte que les lignes insérées par la première transaction reçoivent des valeurs de clé primaire consécutives.

La variable innodb_autoinc_lock_mode contrôle l'algorithme utilisé pour le verrouillage auto-incrémentiel. Elle vous permet de choisir un compromis entre des séquences prévisibles de valeurs d'auto-incrémentation et une concurrence maximale pour les opérations d'insertion.

- **Verrous de prédicats pour les index spatiaux**

InnoDB prend en charge l'indexation SPATIALE des colonnes contenant des données spatiales.

Pour gérer le verrouillage des opérations impliquant des index SPATIAUX, le verrouillage par clé suivante ne fonctionne pas bien pour prendre en charge les niveaux d'isolation de transaction REPEATABLE READ ou SERIALIZABLE. Il n'existe pas de concept d'ordre absolu dans les données multidimensionnelles, de sorte qu'il n'est pas évident de savoir quelle est la clé "suivante".

Pour permettre la prise en charge des niveaux d'isolation pour les tables avec index SPATIAL, InnoDB utilise des verrous prédicats. Un index SPATIAL contient des valeurs de rectangle de délimitation minimal (MBR), de sorte qu'InnoDB assure une lecture cohérente de l'index en définissant un verrou prédicat sur la valeur MBR utilisée pour une requête. Les autres transactions ne peuvent pas insérer ou modifier une ligne qui correspondait à la condition de la requête.

Modèle de transaction InnoDB

- **Niveaux d'isolation des transactions**

L'isolation des transactions est l'un des fondements du traitement des bases de données. L'isolement est le I de l'acronyme ACID ; le niveau d'isolement est le paramètre qui permet d'affiner l'équilibre entre les performances et la fiabilité, la cohérence et la reproductibilité des résultats lorsque plusieurs transactions effectuent des modifications et des requêtes en même temps.

InnoDB offre les quatre niveaux d'isolation des transactions décrits par la norme SQL:1992 : READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ et SERIALIZABLE. Le niveau d'isolation par défaut d'InnoDB est REPEATABLE READ.

Un utilisateur peut modifier le niveau d'isolation pour une session unique ou pour toutes les connexions suivantes à l'aide de l'instruction SET TRANSACTION. Pour définir le niveau d'isolation par défaut du serveur pour toutes les connexions, utilisez l'option --transaction-isolation sur la ligne de commande ou dans un fichier d'options.

InnoDB prend en charge chacun des niveaux d'isolation des transactions décrits ici à l'aide de différentes stratégies de verrouillage. Vous pouvez imposer un haut degré de cohérence avec le niveau REPEATABLE READ par défaut, pour des opérations sur des données cruciales où la conformité ACID est importante. Vous pouvez également assouplir les règles de cohérence avec READ COMMITTED ou même READ UNCOMMITTED, dans des situations telles que le reporting en masse où la cohérence précise et les résultats répétables sont moins importants que la minimisation de la charge de travail pour le verrouillage. SERIALIZABLE applique des règles encore plus strictes que REPEATABLE READ, et est utilisé principalement dans des situations spécialisées, comme avec les transactions XA et pour résoudre des problèmes de concurrence et de blocages.

La liste suivante décrit comment MySQL prend en charge les différents niveaux de transaction. La liste va du niveau le plus couramment utilisé au moins utilisé.

- REPEATABLE READ

Il s'agit du niveau d'isolation par défaut pour InnoDB. Les lectures cohérentes au sein d'une même transaction lisent l'instantané établi par la première lecture. Cela signifie que si vous émettez plusieurs instructions SELECT simples (non verrouillées) dans la même transaction, ces instructions SELECT sont cohérentes les unes par rapport aux autres.

Pour les lectures verrouillées (SELECT avec FOR UPDATE ou FOR SHARE), les instructions UPDATE et DELETE, le verrouillage dépend du fait que l'instruction utilise un index unique avec une condition de recherche unique ou une condition de recherche de type plage.

Pour un index unique avec une condition de recherche unique, InnoDB verrouille uniquement l'enregistrement d'index trouvé, et non l'espace qui le précède.

Pour les autres conditions de recherche, InnoDB verrouille la plage d'index analysée, en utilisant des verrous d'espace ou des verrous de clé suivante pour bloquer les insertions par d'autres sessions dans les espaces couverts par la plage.

- READ COMMITTED

Chaque lecture cohérente, même au sein de la même transaction, définit et lit son propre instantané. Pour les lectures verrouillées (SELECT avec FOR UPDATE ou FOR SHARE), les instructions UPDATE et les instructions DELETE, InnoDB verrouille uniquement les enregistrements de l'index, pas les espaces avant eux, et permet ainsi l'insertion libre de nouveaux enregistrements à côté des enregistrements verrouillés. Le verrouillage des espaces n'est utilisé que pour la vérification des contraintes de clés étrangères et la vérification des clés dupliquées.

Le verrouillage des espaces étant désactivé, des problèmes de lignes fantômes peuvent survenir, car d'autres sessions peuvent insérer de nouvelles lignes dans les espaces.

Seule la journalisation binaire basée sur les lignes est supportée avec le niveau d'isolation READ COMMITTED. Si vous utilisez READ COMMITTED avec binlog_format=MIXED, le serveur utilise automatiquement la journalisation basée sur les lignes.

L'utilisation de READ COMMITTED a des effets supplémentaires :

Pour les instructions UPDATE ou DELETE, InnoDB conserve les verrous uniquement pour les lignes qu'il met à jour ou supprime. Les verrous d'enregistrement pour les lignes non correspondantes sont libérés une fois que MySQL a évalué la condition WHERE. Cela réduit considérablement la probabilité de blocages, mais ils peuvent toujours se produire.

Pour les instructions UPDATE, si une ligne est déjà verrouillée, InnoDB effectue une lecture "semi-consistante", renvoyant la dernière version validée à MySQL afin que ce dernier puisse déterminer si la ligne correspond à la condition WHERE de l'UPDATE. Si la ligne correspond (doit être mise à jour), MySQL lit à nouveau la ligne et, cette fois, InnoDB la verrouille ou attend un verrou.

Considérons l'exemple suivant, en commençant par cette table :

```
CREATE TABLE t (a INT NOT NULL, b INT) ENGINE = InnoDB;

INSERT INTO t VALUES (1,2),(2,3),(3,2),(4,3),(5,2);

COMMIT;
```

Dans ce cas, la table n'a pas d'index, donc les recherches et les balayages d'index utilisent l'index cluster caché pour le verrouillage des enregistrements plutôt que les colonnes indexées.

Supposons qu'une session effectue un UPDATE à l'aide de ces instructions :

Session A

START TRANSACTION;

UPDATE t SET b = 5 WHERE b = 3;

Supposons également qu'une deuxième session effectue une mise à jour en exécutant ces instructions à la suite de celles de la première session :

Session B

UPDATE t SET b = 4 WHERE b = 2;

Lors de l'exécution de chaque UPDATE, InnoDB acquiert d'abord un verrou exclusif pour chaque ligne, puis détermine s'il faut la modifier. Si InnoDB ne modifie pas la ligne, il libère le verrou. Dans le cas contraire, InnoDB conserve le verrou jusqu'à la fin de la transaction. Cela affecte le traitement des transactions de la manière suivante.

Lors de l'utilisation du niveau d'isolation REPEATABLE READ par défaut, le premier UPDATE acquiert un verrou x sur chaque ligne qu'il lit et n'en libère aucun :

x-lock(1,2); retain x-lock

x-lock(2,3); update(2,3) to (2,5); retain x-lock

x-lock(3,2); retain x-lock

x-lock(4,3); update(4,3) to (4,5); retain x-lock

x-lock(5,2); retain x-lock

La deuxième mise à jour se bloque dès qu'elle tente d'acquérir des verrous (parce que la première mise à jour a conservé des verrous sur toutes les lignes) et ne se poursuit pas tant que la première mise à jour n'a pas été validée ou annulée.

x-lock(1,2); block and wait for first UPDATE to commit or roll back

Si READ COMMITTED est utilisé à la place, le premier UPDATE acquiert un x-lock sur chaque ligne qu'il lit et libère ceux des lignes qu'il ne modifie pas :

x-lock(1,2); unlock(1,2)

x-lock(2,3); update(2,3) to (2,5); retain x-lock

x-lock(3,2); unlock(3,2)

x-lock(4,3); update(4,3) to (4,5); retain x-lock

x-lock(5,2); unlock(5,2)

Cependant, si la condition WHERE inclut une colonne indexée et qu'InnoDB utilise l'index, seule la colonne indexée est prise en compte lors de la prise et de la conservation des verrous d'enregistrement. Dans l'exemple suivant, le premier UPDATE prend et conserve un verrou x sur chaque ligne où b = 2. Le second UPDATE se bloque lorsqu'il tente d'acquérir des verrous x sur les mêmes enregistrements, car il utilise également l'index défini sur la colonne b.

CREATE TABLE t (a INT NOT NULL, b INT, c INT, INDEX (b)) ENGINE = InnoDB;

INSERT INTO t VALUES (1,2,3),(2,2,4);

COMMIT;

Session A

START TRANSACTION;

UPDATE t SET b = 3 WHERE b = 2 AND c = 3;

Session B

UPDATE t SET b = 4 WHERE b = 2 AND c = 4;

Le niveau d'isolation READ COMMITTED peut être défini au démarrage ou modifié à l'exécution. Au moment de l'exécution, il peut être défini globalement pour toutes les sessions ou individuellement pour chaque session.

- READ UNCOMMITTED

Les instructions SELECT sont exécutées sans verrouillage, mais une version antérieure possible d'une ligne peut être utilisée. Ainsi, en utilisant ce niveau d'isolation, ces lectures ne sont pas cohérentes. C'est ce qu'on appelle aussi une lecture sale (dirty read). Sinon, ce niveau d'isolation fonctionne comme READ COMMITTED.

- SERIALIZABLE

Ce niveau est similaire à REPEATABLE READ, mais InnoDB convertit implicitement toutes les instructions SELECT simples en SELECT ... FOR SHARE si l'autocommit est désactivé. Si autocommit est activée, le SELECT est sa propre transaction. Il est

donc connu pour être en lecture seule et peut être sérialisé s'il est exécuté en tant que lecture cohérente (non bloquante) et n'a pas besoin de se bloquer pour d'autres transactions. (Pour forcer un simple SELECT à se bloquer si d'autres transactions ont modifié les lignes sélectionnées, il faut désactiver autocommit).

7. Gestion des transactions distribuées :

Dans MySQL, l'architecture des transactions distribuées repose sur le principe de la réplication et la coordination entre les différents nœuds. Cela permet de maintenir l'intégrité des données sur plusieurs sites. MySQL utilise le modèle de réplication maître-esclave, où les transactions sont d'abord appliquées au maître, puis répliquées sur les esclaves.

Une transaction distribuée est un type de transaction de base de données qui s'étend sur plusieurs systèmes distincts dans un réseau. Autrement dit, c'est une transaction de base de données dans laquelle deux hôtes réseau ou plus sont impliqués. Contrairement à une transaction locale qui s'effectue sur une seule base de données, une transaction distribuée implique plusieurs bases de données réparties, qui peuvent être situées sur différents serveurs ou différents emplacements.

L'objectif d'une transaction distribuée est de maintenir la cohérence des données et l'intégrité de la transaction sur tous les systèmes participants, même en cas de défaillance d'un ou plusieurs éléments du réseau. Pour qu'une transaction distribuée soit réussie, toutes les opérations individuelles sur les différents systèmes doivent être complétées avec succès. Si l'une des opérations échoue, la transaction doit être annulée (rollback) sur tous les systèmes pour conserver l'intégrité des données.

De nombreux aspects d'une transaction distribuée sont identiques à ceux d'une transaction dont la portée est une base de données unique. Par exemple, une transaction distribuée fournit un comportement prévisible en appliquant les propriétés ACID qui définissent toutes les transactions.

C'est quoi les propriétés ACID ?

Ces propriétés ACID (Atomique, Cohérent, Isolé et Durable) ont été inventées par les pionniers du traitement des transactions. Pour garantir un comportement prévisible, toutes les transactions doivent posséder ces propriétés de base, ce qui renforce le rôle des transactions critiques en tant que propositions de tout ou rien.

Les caractéristiques de chacune de ces propriété ACID :

- **Atomique** : Une transaction doit s'exécuter exactement une seule fois et doit être atomique : soit tout le travail est fait, soit rien ne l'est. Au sein d'une transaction, les opérations sont interdépendantes et ont très souvent la même intention. En n'effectuant qu'un sous-ensemble de ces opérations, le système pourrait compromettre l'intention globale de la transaction. L'atomicité élimine la possibilité de ne traiter qu'un sous-ensemble d'opérations.
- **Cohérent** : Une transaction doit préserver la cohérence des données, en transformant un état cohérent des données en un autre état cohérent des données. Une grande partie de la responsabilité du maintien de la cohérence incombe au développeur de l'application.
- **Isolé** : Une transaction doit être une unité d'isolement, ce qui signifie que les transactions simultanées doivent se comporter comme si chacune d'entre elles était la seule transaction en cours d'exécution dans le système. Étant donné qu'un degré élevé d'isolement peut limiter le nombre de transactions simultanées, certaines applications réduisent le niveau d'isolement en échange d'un meilleur débit. Pour plus d'informations, reportez-vous à la section Configuration des niveaux d'isolement des transactions.
- **Durable** : Une transaction doit être récupérable et donc pérennité. Si une transaction est validée, le système garantit que ses mises à jour peuvent persister même si l'ordinateur se bloque immédiatement après la validation. La journalisation spécialisée permet à la procédure de redémarrage du système d'effectuer les opérations inachevées requises par la transaction, ce qui rend la transaction durable.

MySQL prend en charge les transactions distribuées entre hôtes via un mécanisme connu sous le nom de XA Transactions, qui permet à plusieurs ressources de données de participer à une transaction globale. Voici les types de transactions distribuées que MySQL peut gérer :

XA Transactions :

- Transactions Globales : Cela commence avec un XA START et se termine par un XA COMMIT ou XA ROLLBACK. Elles englobent plusieurs opérations sur une ou plusieurs ressources de données.
- Transactions Branchées : Elles sont des parties d'une transaction globale, où chaque branche représente la participation d'une ressource de données distincte dans la transaction globale. Cela permet de diviser une transaction globale en plusieurs transactions locales qui sont coordonnées comme une seule transaction.

Transactions à Simple Hôte :

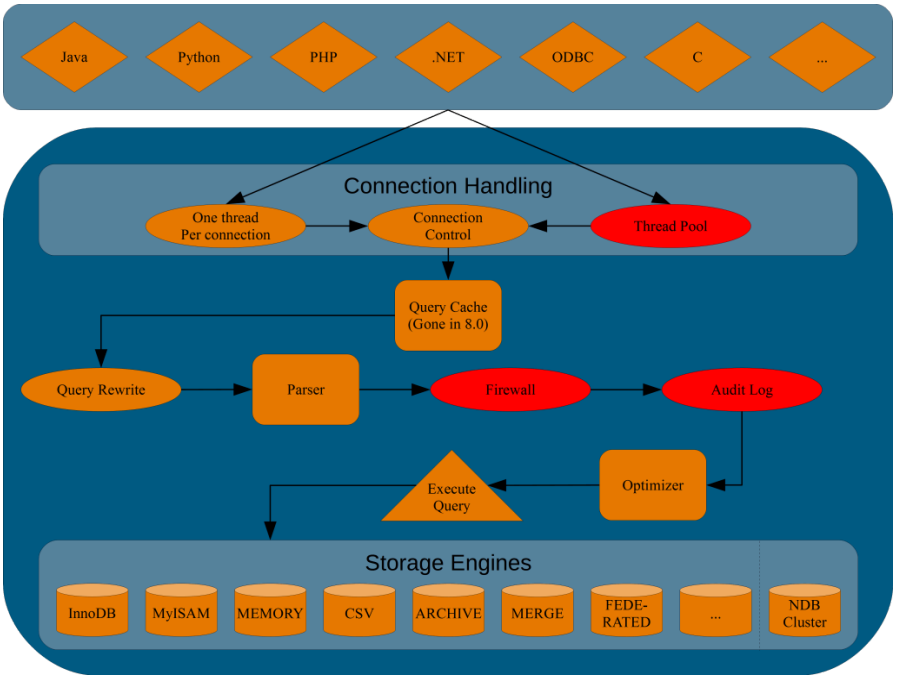
- Ce sont des transactions standard qui s'exécutent sur une seule instance de base de données MySQL. Bien qu'elles ne soient pas distribuées au sens traditionnel, elles peuvent faire partie d'une

opération plus grande lorsqu'elles sont exécutées dans un contexte de transaction distribuée.

MySQL utilise le protocole XA pour coordonner ces transactions distribuées, qui est un standard pour les gestionnaires de transactions distribuées pour interagir avec les gestionnaires de ressources comme les bases de données. Ce protocole garantit que toutes les parties de la transaction distribuée s'engagent à réaliser ou à annuler la transaction de manière cohérente, préservant ainsi l'intégrité des données à travers les systèmes.

MySQL utilise le plugin PECL/mysqlnd_ms pour gérer les transactions distribuées. Le plugin agit comme un coordinateur pour la transaction distribuée, s'assurant que tous les serveurs MySQL participants démarrent une transaction globale. Cette transaction globale peut être validée en utilisant mysqlnd_ms_xa_commit() ou annulée avec mysqlnd_ms_xa_rollback(), ou elle sera implicitement annulée si une erreur survient. Pendant une transaction globale, le plugin surveille les changements de serveurs et s'assure que si une requête est dirigée vers un serveur ne participant pas à la transaction globale, une commande XA BEGIN est émise sur ce serveur avant la requête demandée. Les transactions globales et locales sont mutuellement exclusives dans MySQL, ce qui signifie qu'une transaction globale ne peut pas commencer si une transaction locale est en cours, et la transaction locale doit d'abord être clôturée. De plus, si une tentative est faite pour commencer une transaction XA alors qu'une transaction locale est en cours, le plugin essaie de détecter ce conflit aussi rapidement que possible.

A. Architecture distribuée de MySQL :



La figure ci-dessus montre certaines des fonctionnalités et des plugins de MySQL. La légende orange est disponible à la fois pour la version communautaire et la version commerciale (entreprise), tandis que celle rouge indique que le plugin est exclusif pour la version commerciale. Les éléments ellipsoïdes sont des plugins, tandis que les éléments carrés (oui avec des coins arrondis) indiquent d'autres fonctionnalités. La figure est basée sur MySQL 5.7.20 où le journal d'audit peut être utilisé pour bloquer l'accès aux tables.
En haut se trouvent les différents connecteurs et API qui peuvent être utilisés dans l'application pour se connecter à MySQL. Comme le client en ligne de commande mysql telle que : mysql --user=*user_name* --password *db_name* (voir Commande mysql) utilise l'API C, cela est également inclus dans cette partie de la figure. Il y a plus de connecteurs et d'API qui ne pourraient être visibles sur la figure ; Pour la liste complète, voir la documentation.

La grande boîte sombre représente le serveur. En haut, il y a la gestion des connexions où il y a deux choix : utiliser un thread par connexion ou utiliser le plugin commercial de pool de threads. Un thread par connexion est la façon originale de gérer les nouvelles connexions et, comme son nom l'indique, il s'agit simplement de créer un nouveau thread pour chaque nouvelle connexion. Le pool de threads MySQL Enterprise limitera le nombre total de threads en utilisant à nouveau les threads. Cela conduit à une meilleure évolutivité, en particulier dans le cas de nombreuses connexions exécutant des requêtes courtes. Après la gestion initiale des threads, il est possible d'envoyer la connexion via le plugin optionnel Connection-Control qui limitera les connexions d'un compte, lorsqu'il y a plusieurs tentatives de connexion avec le mauvais mot de passe.

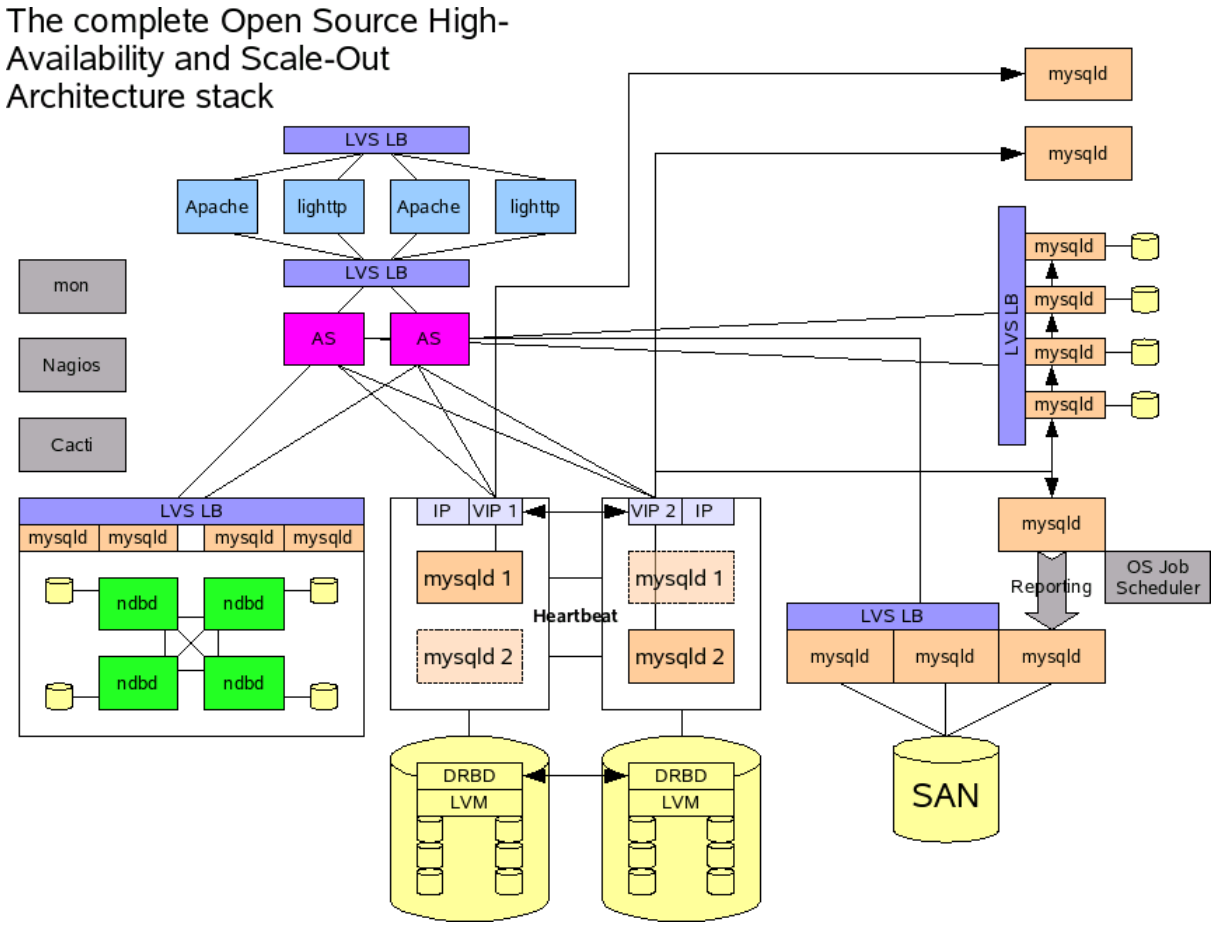
Une fois que la connexion a été effacée pour continuer, la requête est vérifiée par rapport au cache de requête s'il s'agit d'une instruction SELECT et le cache de requête est activé. Pour la plupart des charges de travail, le cache de requête représente une surcharge qui ne justifie pas le gain potentiel. Donc, dans la pratique, il est recommandé de le désactiver complètement.

Par la suite, on a le plug-in facultatif de réécriture des requêtes, qui permet à l'administrateur de base de données de définir les requêtes à réécrire en fonction de requêtes normalisées, comme celles du tableau récapitulatif du résumé du schéma de performance. Par exemple, pour remplacer la requête par – et la même chose pour tous les autres ID, la règle suivante peut être utilisée :

```
SELECT 130 SELECT * FROM world.City WHERE ID = 130
```

```
INSERT INTO query_rewrite.rewrite_rules (pattern, replacement)
VALUES ('SELECT ?', 'SELECT * FROM world.City WHERE ID = ?');
CALL query_rewrite.flush_rewrite_rules();
```

UNE AUTRE IMAGE DE LA PILE COMPLÈTE D'ARCHITECTURE MYSQL OPEN SOURCE À HAUTE DISPONIBILITÉ ET ÉVOLUTIVE :



LÉGENDE :

- DRBD : Périphérique de bloc répliqué distribué
- LVM : Gestionnaire de volumes logiques
- Lighttp : empreinte légère + httpd = LightTPD (prononcé lighty)
- Cactus : Site officiel de Cactus
- Nagios : Nagios est un programme open source de surveillance des hôtes, des services et des réseaux.
- MON : Mon est un planificateur et un outil de gestion des alertes utilisé pour surveiller la disponibilité des services et déclencher des alertes lors de la détection d'une défaillance
- Documentation du cluster MySQL MySQL Cluster
- LVS : Serveur virtuel Linux. Un équilibreur de charge entièrement transparent et à haute disponibilité.

B. Connexion à une base de données distante

MySQL gère la connexion à une base de données distante en utilisant le protocole de communication client-serveur. Voici les étapes générales du processus de connexion à une base de données distante avec MySQL :

- Identification du Serveur MySQL : Lors de la connexion à une base de données distante, le client MySQL doit spécifier l'adresse IP ou le nom d'hôte du serveur MySQL distant auquel il souhaite se connecter.
- Port par défaut :Par défaut, MySQL utilise le port 3306 pour les connexions. Si le serveur MySQL distant utilise un port différent, le client doit spécifier ce port lors de la connexion.
- Authentification : Une fois que la connexion au serveur distant est établie, le processus d'authentification commence. MySQL utilise un système d'authentification basé sur les identifiants (nom d'utilisateur et mot de passe) pour vérifier les droits d'accès.
- Autorisations et Privilèges : Le serveur MySQL distant vérifie les informations d'authentification fournies par le client et accorde ou refuse l'accès en fonction des autorisations et des privilèges définis pour le compte utilisateur.
- Sélection de la Base de Données : Une fois l'authentification réussie, le client peut sélectionner la base de données spécifique sur laquelle il souhaite travailler à l'aide de la commande SQL USE.
- Communication : Une fois la connexion établie et la base de données sélectionnée, le client et le serveur peuvent communiquer en utilisant le protocole MySQL. Les requêtes SQL peuvent être envoyées du client vers le serveur, et les résultats sont renvoyés du serveur vers le client.

Il est important de noter que pour permettre la connexion distante, le serveur MySQL doit être configuré pour accepter les connexions à distance. Cela implique généralement la configuration des autorisations d'accès, la modification des paramètres de sécurité et parfois l'ouverture des ports sur le pare-feu.

Pour des raisons de sécurité, il est recommandé de mettre en œuvre des pratiques telles que l'utilisation de connexions chiffrées (SSL/TLS) et de limiter les droits d'accès aux comptes utilisateur pour minimiser les risques potentiels liés à la connexion distante.

MySQL utilise principalement le protocole TCP/IP pour la connexion à une base de données distante. Cependant, il existe plusieurs méthodes de connexion, chacune ayant son propre protocole ou mécanisme. Voici quelques-uns des protocoles de connexion à une base de données distante avec MySQL :

- TCP/IP (Transmission Control Protocol/Internet Protocol) : C'est le protocole le plus couramment utilisé pour les connexions MySQL à distance.Le client MySQL communique avec le serveur MySQL via TCP/IP, utilisant le port par défaut 3306 (modifiable si nécessaire).
- SSH (Secure Shell) Tunneling : Le protocole SSH peut être utilisé pour créer un tunnel sécurisé entre le client et le serveur MySQL à travers lequel la connexion peut être établie.Cela ajoute une couche de sécurité supplémentaire en chiffrant la communication entre le client et le serveur.
- SSL/TLS (Secure Sockets Layer/Transport Layer Security) : MySQL prend en charge le chiffrement SSL/TLS pour sécuriser les connexions entre le client et le serveur MySQL.Ceci est particulièrement important lorsque la base de données est distante et la communication peut traverser des réseaux non sécurisés.
- Unix Socket : Sur des systèmes d'exploitation de type Unix, MySQL prend en charge la connexion via des sockets Unix locaux.Cela est généralement utilisé pour les connexions locales plutôt que pour les connexions distantes.
- Protocole Pipe (sur Windows) : Sur des systèmes d'exploitation Windows, MySQL prend en charge les connexions via des pipes nommés.Ce protocole est généralement utilisé pour les connexions locales sur des systèmes Windows.

Lors de la configuration de la connexion distante, il est important de considérer la sécurité et d'utiliser des méthodes telles que le chiffrement SSL/TLS ou SSH Tunneling pour protéger la confidentialité des données transitant entre le client et le serveur MySQL. La configuration des paramètres de sécurité sur le serveur MySQL est également essentielle pour garantir une connexion distante sécurisée.

C. Positionnement par rapport aux 12 règles de Date :

Règles	Remarques	Respectée ?
R1 : Autonomie locale ou indépendance du site local	Permet une certaine autonomie locale, mais dans un environnement distribué, certaines dépendances peuvent exister, notamment pour la réplication des données. L'utilisation de *LOAD DATA LOCAL* peut créer des vulnérabilités, notamment en permettant potentiellement à un serveur malveillant d'accéder à des fichiers sur le client. Elle dépend fortement de la configuration correcte par les utilisateurs, ce qui peut limiter l'autonomie dans des environnements moins contrôlés.	Partiellement
R2 : Indépendance du site central	MySQL prend en charge certains aspects de l'indépendance vis-à-vis d'un site central dans la gestion de ses clusters (décentralisés avec les commandes comme *'add hosts', 'remove hosts', 'create site', et 'delete site'*), mais cette indépendance peut être limitée par la nécessité de coordination et de communication entre les nœuds du cluster.	Partiellement
R3 : Fonctionnement continu	MySQL peut être configuré pour permettre des mises à jour continues sans arrêt complet du système, notamment avec des techniques comme la reprise à chaud.	Oui
R4 : Indépendance locale	MySQL prend en charge certains aspects de l'indépendance locale des données, mais cette indépendance peut être limitée par les configurations de sécurité centralisées et la dépendance aux paramètres du serveur.	Partiellement
R5 : Indépendance de la fragmentation	MySQL, grâce à son mécanisme de Group Replication, montre une capacité à gérer l'indépendance de la fragmentation. Cette gestion est réalisée par une fragmentation	Partiellement

	automatique des grands messages et des configurations permettant de définir la taille maximale des messages. Cependant, l'utilisation efficace de cette fonctionnalité dépend de la cohérence des versions et des configurations au sein de l'ensemble du groupe de réplication. Autrement dit, cette indépendance est limitée par la nécessité d'une configuration et d'une compatibilité cohérentes au sein du groupe de réplication.	
R6 : Indépendance de la réplication/duplication	MySQL offre une approche robuste et configurable pour la réplication de groupe, mais avec certaines limitations. Bien qu'il propose des mécanismes avancés pour gérer la taille des transactions (avec des variables comme *'group_replication_transaction_size_limit'* et *'group_replication_communication_max_message_size'* indiquent une gestion efficace de la taille des transactions et des messages, essentielle pour la performance et la stabilité en réplication), la duplication des messages, et la flexibilité des configurations, la nécessité de redémarrages complets pour certains changements et la dépendance à des versions et configurations uniformes peuvent limiter sa flexibilité dans des environnements hétérogènes. MySQL supporte l'indépendance de la duplication/réplication dans une mesure considérable, mais avec des contraintes qui pourraient affecter la gestion et l'adaptabilité dans certains scénarios de déploiement.	Partiellement
R7 : Traitement réparti des requêtes	MySQL peut optimiser localement les requêtes, mais l'optimisation globale dans un environnement distribué peut être limitée.	Partiellement
R8 : Indépendance des transactions distribuées	MySQL offre une conformité élevée à l'indépendance des transactions distribuées. La gestion avancée des transactions, la configuration	Oui

	personnalisable et l'instrumentation détaillée fournissent les outils nécessaires pour maintenir l'indépendance dans un environnement distribué.	
R9 : Indépendance matérielle	MySQL peut fonctionner sur diverses plates-formes matérielles.	Oui
R10 : Indépendance du système d'exploitation	MySQL est compatible avec différents systèmes d'exploitation.	Oui
R11 : Indépendance du réseau	Bien que MySQL ne dépende pas fortement de protocoles réseau spécifiques, la performance et la fiabilité peuvent être affectées par l'infrastructure réseau.	Dans une certaine mesure
R12 : Indépendance de la base de données	MySQL ne prend pas en charge nativement l'interopérabilité avec d'autres SGBD dans un environnement distribué. Il est conçu pour fonctionner de manière homogène avec ses propres instances.	Non

D. Méthode pour garantir l'atomicité, la cohérence et la durabilité :

- **Atomicité** : MySQL utilise le contrôle de transaction pour s'assurer que toutes les opérations d'une transaction sont effectuées ou aucune. Cela est souvent réalisé via des mécanismes de verrouillage et des journaux de transactions.
- **Cohérence** : Les contraintes d'intégrité et les règles métier sont appliquées pour garantir la cohérence des données.
- **Durabilité** : Les modifications effectuées par les transactions sont enregistrées dans des journaux persistants pour garantir qu'elles ne seront pas perdues en cas de panne.
- **Points de contrôle (Checkpoints)** : Les checkpoints périodiques aident à réduire le temps de récupération en mettant à jour l'état actuel du système de fichiers de la base de données.
- **MVCC (Multiversion Concurrency Control)** : Utilisé pour maintenir plusieurs versions d'un enregistrement, ce qui améliore la cohérence et la performance dans un environnement multi-utilisateur.

- **Isolation des transactions** : Les niveaux d'isolation (Read Uncommitted, Read Committed, Repeatable Read, Serializable) permettent de contrôler le degré de visibilité des modifications des transactions en cours.
- **Réplication et Sauvegarde** : La réplication permet de garantir la durabilité et la disponibilité des données. Des sauvegardes régulières sont cruciales pour la récupération après sinistre.
- **Optimisation des performances** : Les index, le partitionnement et le tuning des requêtes contribuent à maintenir la cohérence et l'efficacité des opérations sur les données.

MySQL gère différents niveaux de cohérence (*EVENTUAL, BEFORE_ON_PRIMARY_FAILOVER, BEFORE, AFTER, BEFORE_AND_AFTER*) pouvant être configurés dans la réplication de groupe via la variable *group_replication_consistency*.
Chaque niveau de cohérence offre un compromis entre la disponibilité, la cohérence, et la performance, et peut être sélectionné en fonction des besoins spécifiques de la charge de travail.
Ces niveaux de cohérence affectent le comportement d'un groupe lors de l'élection d'un nouveau primaire, notamment en termes de gestion des transactions en attente et de l'application des transactions en backlog.
Plusieurs scénarios sont possibles pour choisir le niveau de cohérence approprié en fonction des exigences de la charge de travail et des objectifs de performance. Le niveau de cohérence peut être configuré à différents niveaux, y compris à l'échelle de la session ou globalement pour le groupe.
Il est important de choisir le niveau de cohérence approprié en fonction des exigences spécifiques et de la nature des transactions au sein du groupe. (Source)

E. Traitement des pannes dans un environnement distribué :

MySQL gère les pannes dans un environnement distribué en utilisant la réplication pour garantir que les données sont préservées sur plusieurs nœuds. En cas de défaillance d'un nœud, un autre peut prendre le relais, minimisant ainsi l'interruption du service.

Le traitement des pannes dans un environnement distribué pour MySQL implique plusieurs étapes et stratégies pour assurer la continuité et l'intégrité des données.

- **Redondance des Données**
  - Réplication : MySQL utilise la réplication pour créer des copies des données sur plusieurs serveurs. En cas de panne d'un serveur, les autres serveurs répliqués peuvent continuer à servir les données.
  - Types de Réplication : MySQL supporte la réplication synchrone et asynchrone.
  - Group Replication : Une fonctionnalité pour gérer la réplication en groupe, assurant une haute disponibilité.

- **Clustering**
  - MySQL Cluster : Une technologie permettant la distribution des données sur plusieurs nœuds, améliorant la disponibilité et la résilience.
  - Partitionnement automatique : Les données sont réparties automatiquement entre les nœuds.
- **Gestion des Transactions**
  - Transactions Atomiques : S'assurer que les transactions sont complètement effectuées ou annulées en cas de panne.
  - Journalisation : Enregistrer les changements dans un journal pour permettre la récupération après une panne.
- **Détection et Récupération de Pannes**
  - Surveillance : Utilisation d'outils de surveillance pour détecter les pannes rapidement.
  - Recovery : Mécanismes de récupération pour restaurer les données à partir des journaux de transactions ou des sauvegardes.
- **Backup et Restauration**
  - Sauvegardes Régulières : Effectuer des sauvegardes complètes et incrémentielles.
  - Point de Restauration : Créer des points de restauration pour récupérer les données à un état spécifique.
- **Basculement et Équilibrage de Charge**
  - Failover : Passer automatiquement à un serveur de secours en cas de panne.
  - Load Balancing : Répartir la charge entre plusieurs serveurs pour éviter la surcharge et le point unique de défaillance.
- **Mise à Jour et Maintenance**
  - Mise à jour sans Interruption : Appliquer des mises à jour du système sans arrêter le service.
  - Maintenance Proactive : Réaliser une maintenance régulière pour prévenir les pannes.
- **Sécurité et Conformité**
  - Chiffrement : Chiffrer les données pour prévenir les pertes dues à des accès non autorisés.
  - Audit : Suivre les accès et modifications pour assurer la conformité et la sécurité.

Dans un environnement distribué, le traitement des pannes pour MySQL est un processus complexe qui nécessite une planification minutieuse et l'utilisation de diverses techniques et outils pour assurer la haute disponibilité, la redondance, la récupération rapide et la sécurité des données. Il est crucial de mettre en place une stratégie complète couvrant tous les aspects, de la réplication à la surveillance et la maintenance, pour garantir la résilience du système face aux pannes.

8. Gestion de la reprise sur panne :

Une panne est un dysfonctionnement MINEUR ou MAJEUR qui entraîne l'arrêt du fonctionnement total ou partiel du serveur de données. On distingue deux catégories de pannes :

- Les pannes **mineures** : Ce sont des pannes qui concernent la perte des caches mémoire du SGBD. Ou le dysfonctionnement de certains processus.
- Les pannes **majeures** : Ce sont des pannes qui concernent la perte des fichiers contenant les données du SGBD.

En cas de panne : Il faut garantir les propriétés ACID de durabilité et d'atomicité.

- Les données modifiées des transactions COMMITTEES doivent pouvoir, quels que soient la panne, être RECOUVRÉES grâce à la persistance (sauvegarde sur disque) de l'IMAGE APRÈS de ces données
- Les données modifiées des transactions NON COMMITÉES doivent pouvoir, quelques soient la panne, être ANNULÉES grâce à la persistance (sauvegarde sur disque) de l'IMAGE AVANT de ces données

La reprise sur panne dans un cluster de réplication MySQL implique la promotion d'un réplica pour devenir le maître si l'ancien maître échoue. Cela est nécessaire car la réplication MySQL fonctionne comme un collectif de nœuds, où chaque nœud est soit un maître, soit un réplica à tout moment. À partir de MySQL 8.0.22, un mécanisme de basculement de connexion asynchrone a été introduit. Ce mécanisme établit automatiquement une nouvelle connexion de réplication vers une nouvelle source après une défaillance de la connexion d'un réplica à sa source. La commande START REPLICA est utilisée pour tenter de se connecter à une nouvelle source lorsqu'un basculement est initié.

Avec la réplication NDB Cluster, si le processus de réplication primaire du cluster échoue, il est possible de passer à un canal de réplication secondaire en obtenant l'heure du point de contrôle global le plus récent. De plus, MySQL fournit des utilitaires comme mysqlfailover pour la gestion automatique de la reprise sur panne, qui peut être utilisé lorsque la réplication GTID est activée dans MySQL 5.6.

a. Les techniques d’annulation :

Les techniques d'annulation chez MySQL sont principalement liées à la gestion des transactions. Dans un système de gestion de base de données, une transaction est une série d'opérations qui doivent être traitées de manière atomique, c'est-à-dire que soit toutes les opérations réussissent et sont appliquées, soit aucune n'est appliquée. Voici comment fonctionnent les techniques d'annulation dans MySQL :

- **Transactions ACID** : MySQL suit le modèle ACID (Atomicité, Cohérence, Isolation, Durabilité) pour les transactions :
- **Rollback (Annulation)** : En cas d'erreur ou de problème pendant une transaction, MySQL utilise l'opération ROLLBACK pour annuler toutes les modifications effectuées par la transaction en cours. Cette opération ramène la base de données à son état avant le début de la transaction.
- **Journalisation des Transactions** : MySQL utilise un journal de transactions pour enregistrer les modifications apportées par chaque transaction. Si le système échoue avant la fin d'une transaction, MySQL utilise le journal pour annuler (rollback) les modifications non validées lors du redémarrage.
- **Points de Sauvegarde (Savepoints)** : MySQL permet de créer des points de sauvegarde à l'intérieur d'une transaction. Avec SAVEPOINT, on peut définir un point dans la transaction où on peut revenir plus tard sans annuler toute la transaction. ROLLBACK TO SAVEPOINT permet d'annuler une partie de la transaction depuis le dernier point de sauvegarde.
- **Verrouillage Optimiste et Pessimiste** :
    - Verrouillage pessimiste : MySQL verrouille les données lors de leur lecture pour empêcher d'autres transactions de les modifier jusqu'à la fin de la transaction en cours.
    - Verrouillage Optimiste : Les conflits sont gérés au moment de la validation de la transaction, idéal pour les environnements avec moins de conflits.
- **Gestion des Deadlocks** : MySQL détecte les situations de deadlock et annule automatiquement une des transactions impliquées pour résoudre le blocage.

Les techniques d'annulation dans MySQL sont conçues pour assurer l'intégrité et la cohérence des données dans un environnement multi-utilisateur et pour gérer efficacement les erreurs et les conflits qui peuvent survenir pendant les opérations de la base de données. Elles sont essentielles pour maintenir la fiabilité du système de gestion de base de données.

b. La journalisation :

La journalisation des transactions dans MySQL, particulièrement avec le moteur de stockage InnoDB, est un aspect crucial pour garantir l'intégrité des données et la récupération en cas de panne. Cette fonctionnalité joue un rôle essentiel dans le maintien de la propriété ACID (Atomicité, Cohérence, Isolation, Durabilité) des transactions.

a. Objectif de la Journalisation des Transactions :

La journalisation des transactions sert à enregistrer toutes les modifications apportées à la base de données dans un format sécurisé et séquentiel. En cas de panne système, ces journaux permettent de restaurer la base de données à un état cohérent, soit en refaisant (redoing) les transactions complétées, soit en annulant (undoing) les transactions incomplètes.

b. Composants de Journalisation

Redo Log : Enregistre toutes les modifications des données. C'est un mécanisme de journalisation en avant (forward logging).
Undo Log : Enregistre les informations nécessaires pour annuler une transaction. C'est une forme de journalisation en arrière (backward logging).

c. Fonctionnement de la Journalisation

Lorsqu'une transaction est initiée, les modifications des données ne sont pas immédiatement écrites sur le disque mais sont d'abord stockées dans la mémoire tampon (buffer pool). En parallèle, les détails de la transaction sont enregistrés dans le redo log buffer. Périodiquement, le contenu du redo log buffer est écrit de manière séquentielle dans les fichiers redo log sur le disque. Cela se produit lors des événements de commit, mais aussi à d'autres moments en fonction de la configuration et de la charge du système.

d. Processus de Commit

Lorsqu'une transaction est validée (commit), MySQL s'assure que les entrées correspondantes dans le redo log sont durables sur le disque avant de confirmer la réussite de la transaction. Cela garantit la propriété de durabilité des transactions : même en cas de panne, l'état de la transaction peut être récupéré.

e. Récupération après Panne

En cas de redémarrage après un crash, MySQL utilise les redo logs pour refaire (replay) les transactions qui ont été validées mais dont les modifications n'ont pas été complètement écrites sur le disque. Les undo logs sont utilisés pour annuler toutes les transactions qui étaient en cours et n'ont pas été validées avant la panne.

f. Gestion des Espaces de Stockage

Les redo logs sont stockés dans un espace de stockage fixe, généralement configuré lors de l'installation de la base de données.
Les fichiers redo log fonctionnent en rotation : une fois qu'un fichier est rempli, MySQL passe au fichier suivant.

La journalisation des transactions est un mécanisme essentiel pour garantir la fiabilité et la robustesse de MySQL. Elle permet non seulement de préserver l'intégrité des données en cas de panne mais aussi d'assurer que les transactions respectent les principes ACID. Cette fonctionnalité est un composant clé qui fait de MySQL une solution de base de données fiable pour des applications critiques.

g. L'utilisation de SET TRANSACTION

L'utilisation de SET TRANSACTION dans MySQL est un outil important pour contrôler les caractéristiques des transactions.

```
SET [GLOBAL | SESSION] TRANSACTION
    transaction_characteristic [, transaction_characteristic] ...

transaction_characteristic: {
    ISOLATION LEVEL level
  | access_mode
}

level: {
     REPEATABLE READ
   | READ COMMITTED
   | READ UNCOMMITTED
   | SERIALIZABLE
}

access_mode: {
     READ WRITE
   | READ ONLY
}
```

Table 13.9 SET TRANSACTION Syntax for Transaction Characteristics

Syntax	Affected Characteristic Scope
`SET GLOBAL TRANSACTION transaction_characteristic`	Global
`SET SESSION TRANSACTION transaction_characteristic`	Session
`SET TRANSACTION transaction_characteristic`	Next transaction only

Table 13.10 SET Syntax for Transaction Characteristics

Syntax	Affected Characteristic Scope
`SET GLOBAL var_name = value`	Global
`SET @@GLOBAL.var_name = value`	Global
`SET PERSIST var_name = value`	Global
`SET @@PERSIST.var_name = value`	Global
`SET PERSIST_ONLY var_name = value`	No runtime effect
`SET @@PERSIST_ONLY.var_name = value`	No runtime effect
`SET SESSION var_name = value`	Session
`SET @@SESSION.var_name = value`	Session
`SET var_name = value`	Session
`SET @@var_name = value`	Next transaction only

Il est possible de vérifier les valeurs globales et de session de Caractéristiques de la transaction au moment de l'exécution :

```
SELECT @@GLOBAL.transaction_isolation, @@GLOBAL.transaction_read_only;
SELECT @@SESSION.transaction_isolation, @@SESSION.transaction_read_only;
```

Niveaux d'isolement des transactions :

Le niveau d'isolement d'une transaction détermine dans quelle mesure la transaction est isolée des modifications apportées par d'autres transactions. Ceci est crucial pour prévenir les problèmes comme les lectures sales, les lectures non répétables, et les lectures fantômes.

Options de Niveau d'Isolement :

- REPEATABLE READ (par défaut) : Assure que les résultats de la lecture sont répétables au sein de la même transaction.
- READ COMMITTED : Autorise la lecture des lignes qui ont été validées (committed).
- READ UNCOMMITTED : Permet de lire les modifications non validées d'autres transactions.
- SERIALIZABLE : Le niveau le plus strict, assurant que les transactions sont traitées de manière séquentielle.

Mode d'accès aux transactions :

Détermine si une transaction peut effectuer des écritures (modifications) ou est limitée à des lectures.
Options de Mode d'Accès :
READ WRITE : Autorise les lectures et écritures dans la transaction (par défaut).
READ ONLY : Limite la transaction à des opérations de lecture, ce qui peut améliorer les performances dans certains cas.

Portée des caractéristiques de transaction :

Options de Portée :
GLOBAL : Applique les caractéristiques à toutes les sessions à venir.
SESSION : Applique les caractéristiques à toutes les transactions subséquentes dans la session courante.
Aucun mot-clé : Applique les caractéristiques uniquement à la transaction suivante.

Erreurs et Contraintes :

Les caractéristiques de transaction ne peuvent pas être modifiées pendant qu'une transaction est en cours. La modification des caractéristiques globales nécessite des privilèges spéciaux (*CONNECTION_ADMIN ou SUPER*).

Configuration au démarrage :

Utiliser --transaction-isolation=level et --transaction-read-only dans la ligne de commande ou le fichier de configuration pour définir le niveau d'isolement et le mode d'accès par défaut au démarrage du serveur.

Vérification des valeurs :

Il est possible de vérifier les valeurs de configuration actuelles à l'aide de requêtes SELECT.

L'instruction SET TRANSACTION dans MySQL permet aux administrateurs de bases de données de contrôler finement le comportement des transactions, en termes d'isolement et de mode d'accès, pour optimiser les performances et garantir l'intégrité des données. Ces réglages sont essentiels pour gérer les conflits entre transactions et assurer la cohérence des données dans un environnement multi-utilisateur.

c. Les sauvegardes :

Les sauvegardes dans MySQL est une stratégie essentielle pour assurer la durabilité et la disponibilité des données. Elle implique la création et l'utilisation de sauvegardes pour restaurer les données après un crash ou une corruption des données.

a. Types de Sauvegardes

Sauvegardes Complètes : Copie de l'ensemble de la base de données. Elles servent de point de départ pour la récupération après sinistre.
mysqldump : Outil le plus courant pour créer une sauvegarde complète.
mysqldump -u [username] -p [database_name] > backup_file.sql
[username] : Nom d'utilisateur pour se connecter à MySQL.
[database_name] : Nom de la base de données à sauvegarder.
backup_file.sql : Fichier de sortie contenant la sauvegarde.
Sauvegardes Incrémentielles : Enregistrent uniquement les changements effectués depuis la dernière sauvegarde (complète ou incrémentielle). Cela réduit l'espace de stockage et le temps nécessaire pour effectuer des sauvegardes régulières.
Percona XtraBackup :

- Première sauvegarde (complète) :

```
xtrabackup --backup --target-dir=/path/to/full/backup
```

- Sauvegardes incrémentielles :

```
xtrabackup --backup --target-dir=/path/to/inc/backup --incremental-basedir=/path/to/full/backup
```

Sauvegardes Binaires (Binary Log) : Enregistrent toutes les modifications apportées à la base de données. Ces logs sont essentiels pour la reprise après sinistre car ils permettent de rejouer les transactions après la restauration d'une sauvegarde complète ou incrémentielle.

- Activer les Logs Binaires :
  - Configurez my.cnf (ou my.ini sur Windows) :

    ```
    [mysqld]
    log-bin=mysql-bin
    server-id=1
    ```

  - Redémarrez le serveur MySQL après modification.
- Sauvegarde des Logs Binaires : Utilisez mysqldump avec l'option --master-data pour inclure les informations de position du log binaire.

  ```
  mysqldump --master-data=2 -u [username] -p [database_name] > backup_file.sql
  ```

b. Automatisation et Planification des Sauvegardes

Planification : Les sauvegardes doivent être planifiées régulièrement pour minimiser la perte de données en cas de panne.
Automatisation : L'utilisation d'outils et de scripts pour automatiser les processus de sauvegarde et de restauration peut réduire les erreurs humaines et garantir la régularité des sauvegardes.
Utilisez des outils comme cron sous Linux pour planifier des sauvegardes régulières.
Par exemple, un script cron pour une sauvegarde quotidienne à minuit :
0 0 * * * /usr/bin/mysqldump -u [username] -p[password] [database_name] > /path/to/

c. Procédure de Restauration

Restauration à partir d'une Sauvegarde Complète : Première étape pour récupérer les données jusqu'au point de la sauvegarde.
Application des Sauvegardes Incrémentielles : Après la restauration d'une sauvegarde complète, appliquer les sauvegardes incrémentielles pour récupérer les données jusqu'au point de la dernière sauvegarde incrémentielle.
Utilisation des Logs Binaires : Pour récupérer les données jusqu'au moment de la panne, appliquer les transactions enregistrées dans les logs binaires après la dernière sauvegarde incrémentielle..

d. Utilisation des GTID (Global Transaction Identifiers)

Utilisation des GTID pour la Synchronisation : Les GTID aident à identifier les transactions manquantes dans le membre rejoignant le groupe. Ils ne marquent pas un point précis dans le temps pour la synchronisation ni ne transmettent d'informations sur la certification.
Rôle des Marqueurs de Vue : Les marqueurs de vue dans les journaux binaires indiquent des changements dans la composition du groupe.
Ils contiennent des métadonnées supplémentaires pour aider à la synchronisation et à la certification.
Changements de Vue : Une vue représente un groupe de membres actifs à un moment donné. Un changement de vue se produit lors de l'ajout ou du retrait de membres.
Chaque changement génère un identifiant de vue unique.
Identificateur de Vue : Composé de deux parties : une partie générée aléatoirement et un entier croissant. L'entier est incrémenté à chaque changement de vue.
Processus de Récupération Distribuée :

- Début avec un groupe stable : Tous les serveurs traitent les transactions.

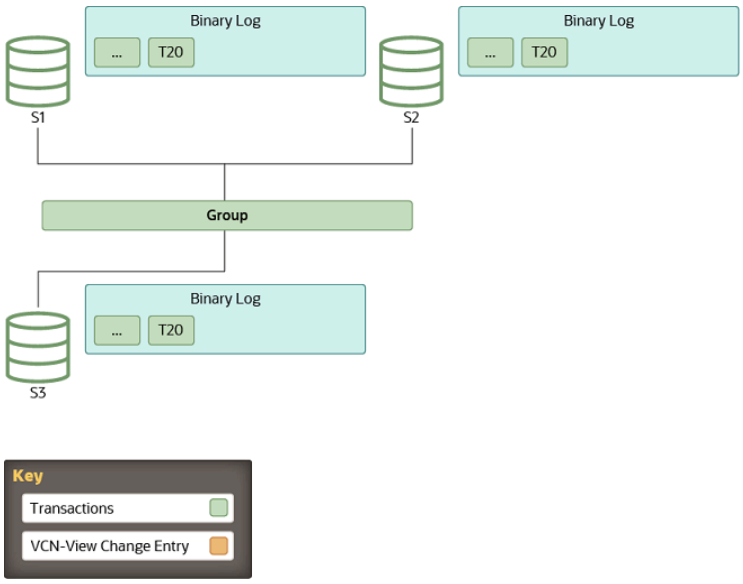


Figure : Groupe Stable

- Adhésion d'un Membre : Quand un nouveau membre rejoint, un événement de changement de vue est enregistré dans les journaux binaires des serveurs existants.

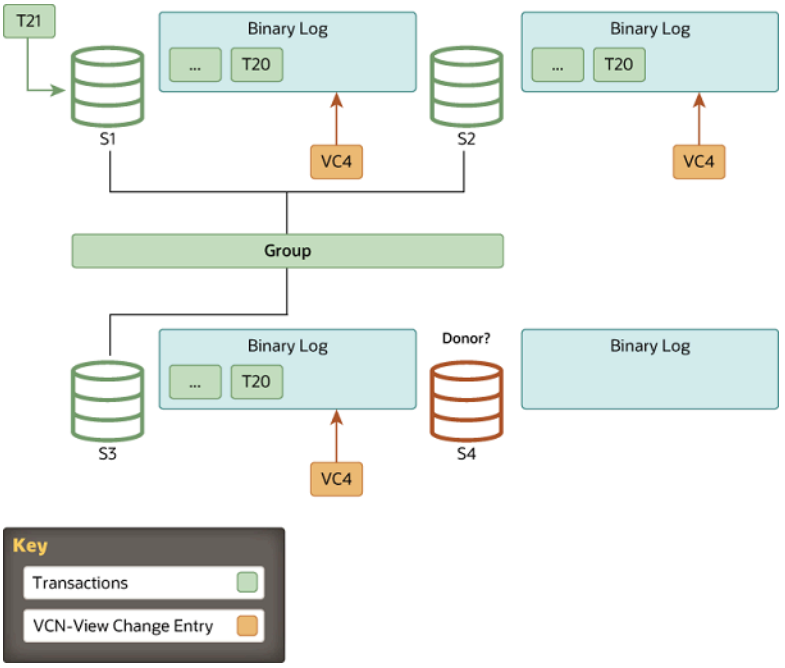


Figure : adhésion d'un membre

- Transfert d'État et rattrapage : Le membre rejoignant effectue un transfert d'état à partir du journal binaire d'un donneur choisi. Si un grand écart de transactions existe, une opération de clonage peut être initiée.

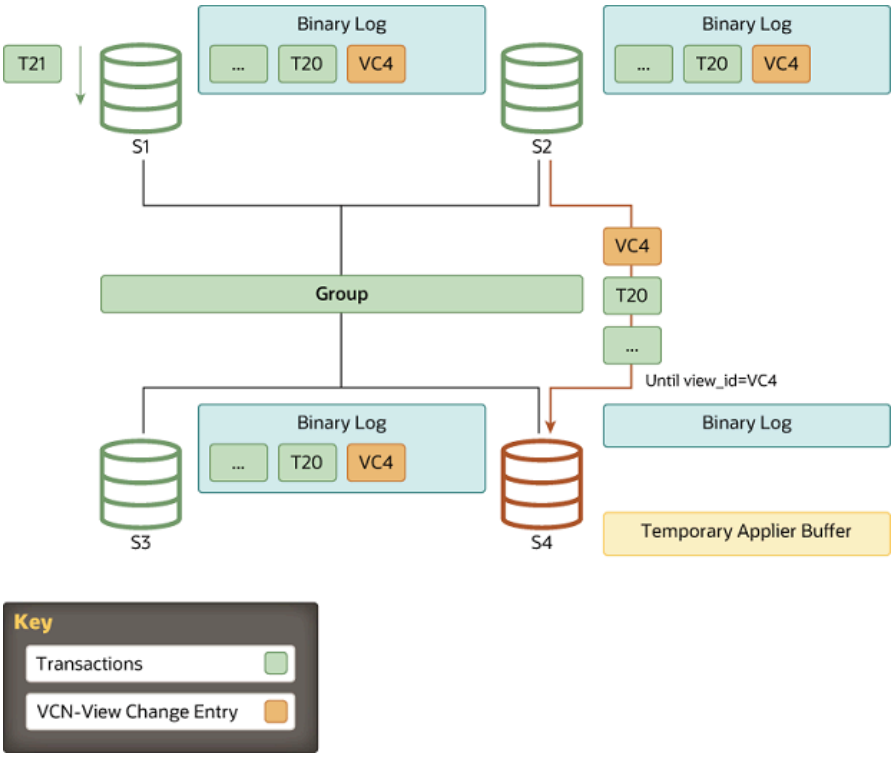
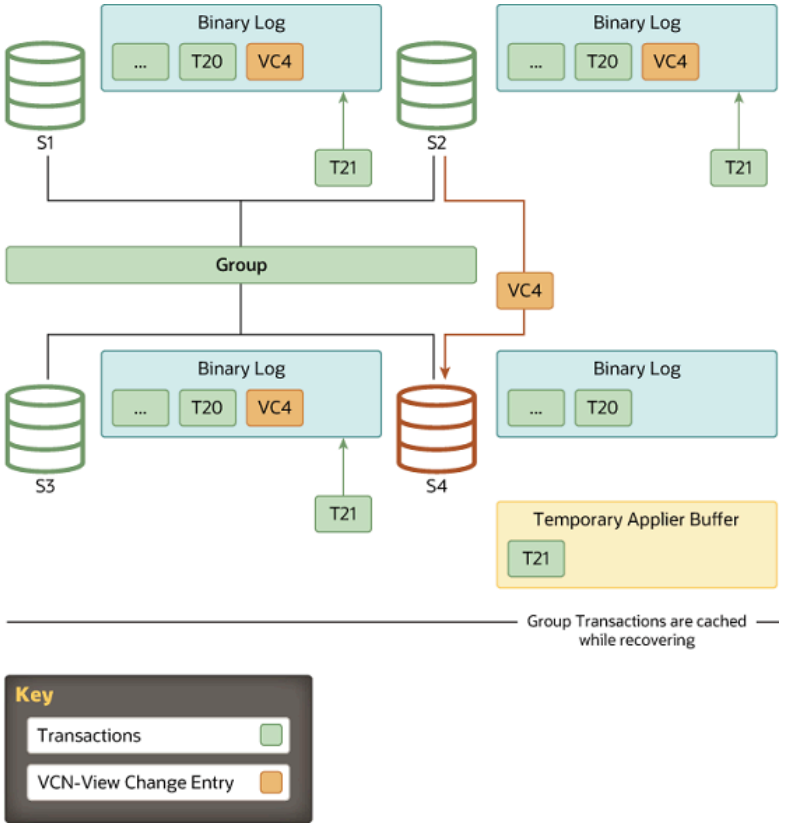


Figure : transfert d'Etat : rattrapage

- Application des Transactions : Après avoir atteint le point de synchronisation marqué par l'identifiant de vue, le nouveau membre applique les transactions mises en cache pour se synchroniser avec les autres membres du groupe.



Transfert en file d'attente

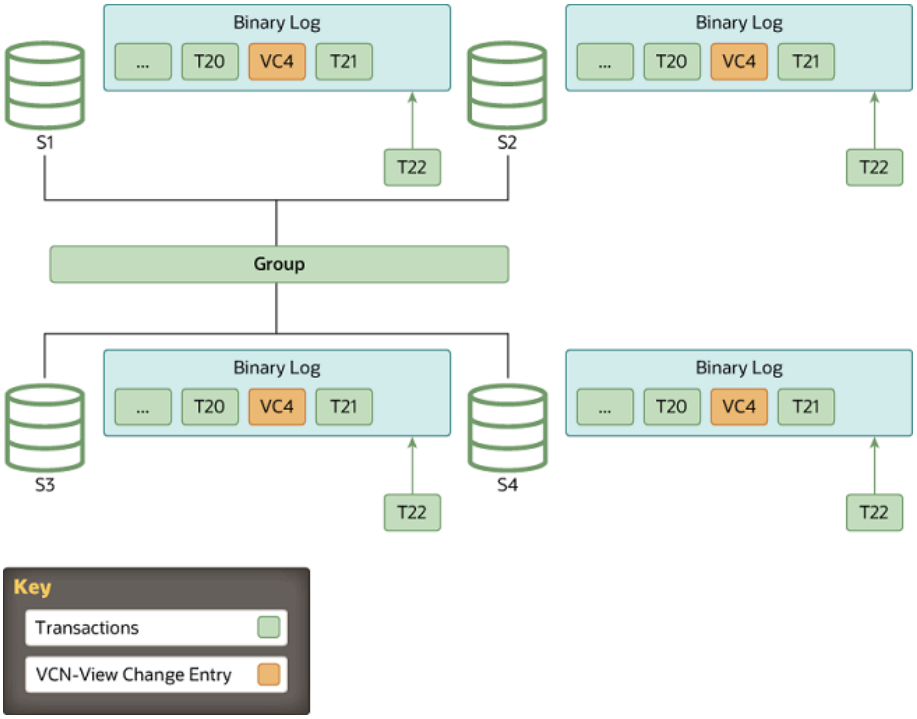


Figure : transfert en ligne

Rôle du Journal des Modifications de Vue : Sert de marqueur dans le journal binaire pour délimiter les changements de vue. Transmet des informations de certification nécessaires pour les transactions futures.
Fin du Processus : Une fois que le nouveau membre a rattrapé son retard et synchronisé ses données avec les autres membres, il devient un membre en ligne actif du groupe.

En résumé, la récupération distribuée dans la réplication de groupe MySQL utilise les GTID pour identifier les transactions manquantes et les marqueurs de vue pour marquer les changements de configuration du groupe. Ce processus permet aux nouveaux membres de se synchroniser efficacement avec le groupe existant, en garantissant l'intégrité et la cohérence des données au sein du groupe. (Source)

d. Gestion de la reprise à chaud :

La gestion de la reprise à chaud chez MySQL, aussi connue sous le terme de "Hot Backup", désigne un mécanisme qui permet de créer une sauvegarde d'une base de données tout en la maintenant en fonctionnement.

- **Non-Interruption des opérations** : La reprise à chaud permet de créer des sauvegardes sans arrêter ou ralentir significativement les opérations de la base de données. Cela signifie que les transactions peuvent continuer à s'exécuter pendant la sauvegarde.

- **Consistance des données** : L'un des défis majeurs de la reprise à chaud est de garantir la consistance des données. MySQL doit s'assurer que la sauvegarde reflète un état cohérent de la base de données, même si des transactions sont en cours pendant la sauvegarde.

- **Journalisation des transactions (Log)** : MySQL utilise un système de journalisation des transactions pour soutenir la reprise à chaud. Les changements sont d'abord enregistrés dans un journal de transactions avant d'être appliqués à la base de données. En cas de panne, ces journaux sont utilisés pour restaurer les transactions qui étaient en cours au moment de la sauvegarde.

- **Utilisation des snapshots** : Certains systèmes de reprise à chaud utilisent des snapshots (instantanés) du système de fichiers pour capturer l'état de la base de données à un moment précis, tout en permettant la poursuite des opérations normales.

- **Outils de sauvegarde** : MySQL offre des outils comme MySQL Enterprise Backup, qui permettent de réaliser des sauvegardes à chaud. Ces outils sont conçus pour interagir efficacement avec le serveur MySQL, en minimisant l'impact sur les performances.

- **Restauration rapide** : En plus de créer des sauvegardes, la gestion de la reprise à chaud comprend aussi des mécanismes pour restaurer rapidement les données à partir de ces sauvegardes en cas de panne ou de corruption des données.

- **Point-in-Time Recovery (PITR)** : Il s'agit d'une fonctionnalité permettant de restaurer la base de données à un état précis dans le temps, utilisant les journaux de transactions en plus de la sauvegarde.

La gestion de la reprise à chaud chez MySQL est essentielle pour assurer la continuité des opérations et la protection des données, en permettant de réaliser des sauvegardes sans interrompre le service et en garantissant la consistance des données sauvegardées.

e. Gestion de la reprise à froid :

La gestion de la reprise à froid dans MySQL, également connue sous le terme de "Cold Backup", se réfère à un processus de sauvegarde où le serveur de base de données est complètement arrêté avant de réaliser la sauvegarde.

- **Arrêt du serveur** : Pour effectuer une sauvegarde à froid, le serveur MySQL doit être complètement arrêté. Cela signifie qu'aucune transaction ne peut s'exécuter pendant la sauvegarde.

- **Intégrité des données** : L'un des principaux avantages de la sauvegarde à froid est qu'elle garantit l'intégrité des données. Comme il n'y a pas de transactions en cours, le risque d'incohérence dans les données sauvegardées est nul.

- **Simplicité du processus** : La sauvegarde à froid est généralement plus simple à mettre en œuvre que la sauvegarde à chaud, car elle ne nécessite pas de mécanismes complexes pour gérer les transactions en cours. Il s'agit souvent de copier simplement les fichiers de données du serveur MySQL.

- **Impact sur la Disponibilité** : Le principal inconvénient de la sauvegarde à froid est qu'elle nécessite une interruption complète du service. Cela peut ne pas être acceptable pour les bases de données devant être disponibles en permanence.

- **Restauration** : La restauration à partir d'une sauvegarde à froid est généralement directe. Une fois que les fichiers de données sont restaurés, le serveur peut être redémarré normalement.

- **Utilisation dans les scénarios de test ou de maintenance** : La reprise à froid est souvent utilisée dans des scénarios où la base de données peut être mise hors ligne sans impact majeur, comme pour des tests ou des opérations de maintenance.

- **Considérations de sécurité** : Les sauvegardes à froid sont souvent considérées comme plus sûres du point de vue de la sécurité, car elles évitent les complications liées à l'interaction avec un serveur actif.

La gestion de la reprise à froid chez MySQL est une approche de sauvegarde qui privilégie la simplicité et l'intégrité des données au détriment de la disponibilité. Elle est bien adaptée aux environnements où il est acceptable de mettre la base de données hors ligne pendant la sauvegarde. (Source)

9. Techniques d'indexation :

MySQL utilise les index pour localiser rapidement des lignes avec des valeurs de colonne spécifiques. Sans index, MySQL doit lire toute la table pour trouver les lignes pertinentes.

La création et la gestion des index permet d'accélérer l'accès aux données lors des requêtes de recherche. Mais ils peuvent augmenter le temps nécessaire pour les opérations de modification de données (ajouts, suppressions et mises à jour). Ceci est dû au fait que chaque modification de données nécessite une mise à jour de l'index, ce qui peut affecter les performances générales du système.

Dans MySQL, un index est constitué de plusieurs composants clés :

- **Colonnes :** Ce sont les colonnes d'une table sur lesquelles l'index est construit. Chaque entrée dans ces colonnes d'index est liée à la clé primaire de la ligne correspondante dans la table, formant une association index / ligne.
- **Clé :** La clé d'index est une structure de données organisée, qui intègre les valeurs des colonnes d'index et des pointeurs vers les lignes pertinentes de la table. Cela permet un accès rapide et efficace aux lignes de la table basé sur les valeurs des colonnes d'index.

Création d'index :

MySQL offre deux méthodes principales pour créer des index :

- **CREATE [UNIQUE] INDEX *nomIndex***
- **ALTER TABLE ... ADD INDEX *nomIndex***

La commande ***CREATE INDEX nomIndex ON table (colonne1, colonne2, ...);*** est utilisée exclusivement pour ajouter un nouvel index à une table existante. Cette commande sert à créer un index sans modifier d'autres aspects de la structure de la table. Elle permet d'indexer une ou plusieurs colonnes, et l'ordre de ces colonnes dans la définition de l'index peut influencer l'efficacité de l'index pour certaines requêtes.

La commande ***ALTER TABLE table ADD INDEX nomIndex (colonne);*** fait partie de la commande ALTER TABLE et est utilisée pour modifier la structure de la table en y ajoutant un index. Cette méthode est utile lorsqu'on souhaite apporter plusieurs modifications à une table, y compris l'ajout d'index, car elle permet de regrouper différentes modifications dans une seule et même commande.

Le choix entre **CREATE INDEX** et **ALTER TABLE ... ADD INDEX** dépend des besoins et du contexte dans lequel l'index est créé.

Différents types d'Index :

Index B-Tree

Les index B-Tree sont les plus couramment utilisés dans MySQL. Ils sont utilisés pour des requêtes de recherche, de tri et d'agrégation.

Par exemple, les nœuds feuilles d'un index B-Tree contiennent des pointeurs vers les lignes, mais ils contiennent également les données de la colonne indexée elle-même. Cela permet de créer des index couvrants qui éliminent le besoin de recherches supplémentaires dans la table.

Si nous avons une table **pilotes** avec des colonnes **nom** et **email**. Pour créer un index B-Tree sur la colonne **email**, vous pouvez utiliser la commande suivante :

```
CREATE INDEX index_email ON pilotes (email);
```

Index FULLTEXT

Ce type d'index est utilisé pour les recherches de texte. Ils fonctionnent différemment des index B-Tree car ils ne comparent pas les valeurs mais effectuent plutôt une recherche par mots-clés. Les index FULLTEXT ne sont pas utilisés pour les conditions WHERE mais plutôt avec l'opérateur MATCH AGAINST.

Si nous avons une table **pilotes** avec une colonne **nom** de type TEXT. Créer un index FULLTEXT facilitera la recherche de mots spécifiques dans le contenu :

```
CREATE FULLTEXT INDEX index_nom ON pilotes (nom);
```

Avec cet index, on peut exécuter des recherches de FULLTEXT avec la syntaxe MATCH...AGAINST.

```
SELECT * FROM pilotes WHERE MATCH (nom) AGAINST ('nom_pilotes');
```

Bonnes Pratiques pour l'Indexation :

Ordre des Colonnes dans les Index Composites :

L'efficacité d'un index dépend de l'ordre dans lequel les colonnes sont indexées. L'ordre doit correspondre à la manière dont les colonnes sont utilisées dans les requêtes. Par exemple, si une requête filtre d'abord par la colonne A puis par la colonne B, un index sur (A, B) sera plus efficace qu'un index sur (B, A).

Taille des Index :

La limitation de la taille des index est importante dans les grandes bases de données. Des index trop volumineux peuvent être coûteux à maintenir et à mettre à jour, en particulier lors des insertions, mises à jour, et suppressions de données.

Une taille d'index limitée (maximum 5 colonnes) permet de maintenir un équilibre entre les avantages en termes de vitesse de lecture et les coûts liés à la mise à jour des index.

Index Fonctionnels :

À partir de MySQL 8.0, la prise en charge des index fonctionnels a été introduite. Ces index permettent d'indexer les résultats des fonctions appliquées aux colonnes, ce qui est utile pour des requêtes impliquant des calculs ou des manipulations de données complexes.

Les index fonctionnels offrent une flexibilité accrue pour l'optimisation des requêtes, en permettant l'indexation de valeurs transformées ou calculées, ce qui n'était pas possible avec les index traditionnels.

Par exemple, si nous avons une table **vol** avec une colonne **date_vol** de type **DATETIME** et que l'on souhaite optimiser les requêtes qui cherchent des commandes en fonction du mois de la date de commande. Au lieu d'indexer

directement la colonne date_commande, on peut créer un index fonctionnel sur le mois extrait de cette date :

```
ALTER TABLE vol ADD INDEX index_mois_vol ((MONTH(date_vol)));
```

10.Optimisation de requêtes :

Dans MySQL, l'optimisation de requêtes est essentielle pour améliorer les performances et l'efficacité de la base de données. Cette optimisation comprend l'utilisation de l'optimiseur de requêtes de MySQL, la structuration efficace des requêtes SQL, et l'exploitation des index.

Les moyens d'optimiser les requêtes :

L'optimiseur de requêtes de MySQL joue un rôle essentiel dans l'interprétation et l'exécution des requêtes SQL. Il analyse les requêtes, choisit les plans d'exécution, et utilise les index pour améliorer la vitesse d'exécution.

L'optimiseur prend en compte plusieurs facteurs, tels que :

- les statistiques de la table,
- la structure des index,
- la cardinalité des colonnes.

Pour Optimisation des requêtes InnoDB, le manuel de référence MySQL 8.0 fournit des conseils pour optimiser les requêtes sur les tables InnoDB. Voici quelques un :

★ Création d'index : Il est essentiel de créer un ensemble approprié d'index sur chaque table pour améliorer les performances des requêtes InnoDB.
  - Exemple :
    Supposons que nous avons une table pilotes avec une colonne id_pilote et une table vols avec une colonne pilote_id. Un index sur pilote_id dans vols améliorerait les performances des requêtes jointes :
    ```
    SELECT * FROM pilotes p JOIN vols v ON p.id_pilote = v.pilote_id;
    ```

★ Clés primaires et secondaires : Chaque table InnoDB a une clé primaire, qu'elle soit explicitement définie ou non. Il est recommandé de spécifier des colonnes de clé primaire utilisées dans les requêtes les plus importantes et critiques en termes de temps. Éviter de spécifier trop de colonnes ou des colonnes trop longues dans la clé primaire, car cela peut réduire les performances.

★ Index secondaires et index concaténés : Il n'est pas conseillé de créer un index secondaire pour chaque colonne. Au lieu de cela, envisagez de créer un petit nombre d'index concaténés pour couvrir différentes combinaisons de colonnes testées par plusieurs requêtes.

★ Optimisation des colonnes indexées : Si une colonne indexée ne peut pas contenir de valeurs NULL, déclarez-la comme NOT NULL. Cela aide l'optimiseur à déterminer plus efficacement l'index le plus adapté pour une requête.

★ Éviter les Colonnes Inutiles : Au lieu de sélectionner toutes les colonnes avec SELECT *, spécifiez uniquement les colonnes nécessaires.

★ Transactions en lecture seule : Il est possible d'optimiser les transactions en lecture seule pour les tables InnoDB en utilisant des techniques spécifiques.

Améliorer la performance :
Les aspects suivants permettent d'améliorer les performances des requêtes :

➢ Utilisation de la fonction *EXPLAIN* : Cet outil aide à comprendre comment MySQL exécute une requête, ce qui permet d'identifier les points à optimiser.
On l'ajoute avant la requête SQL. Cela fournit des détails sur la manière dont MySQL exécute la requête, y compris les informations sur les index utilisés, le nombre de lignes examinées, et les différentes étapes de l'exécution.
Voici un exemple simple :

```
EXPLAIN SELECT * FROM votre_table;
```

Cela affichera un tableau avec des colonnes comme id, select_type, table, type, possible_keys, key, key_len, ref, rows, filtered, et Extra. Chacune de ces colonnes donne des indications sur la façon dont MySQL traite la requête, vous aidant ainsi à identifier les points à optimiser.

➢ Stratégies d'accès aux tables : Différentes méthodes d'accès aux tables peuvent impacter les performances, notamment le choix entre un balayage complet de table et l'utilisation d'index.

➢ Gestion de la mémoire tampon : Il est important de configurer correctement la taille et l'utilisation du tampon pour optimiser l'accès aux données et aux index.
Les paramètres clés incluent '*innodb_buffer_pool_size*', qui détermine la taille du pool de mémoire tampon pour InnoDB, et '*innodb_buffer_pool_instances*', qui définit le nombre d'instances du pool de mémoire tampon. Ajuster ces paramètres en fonction de la quantité de mémoire disponible et des besoins

de votre base de données peut améliorer significativement les performances en réduisant les opérations d'E/S disque.

➢ Optimisation des jointures : Les jointures entre les tables doivent être conçues de manière efficace pour réduire les coûts en termes de temps d'exécution.

La section "Optimisation des requêtes avec LIMIT" du manuel MySQL 8.0 traite de l'optimisation des performances des requêtes utilisant la clause LIMIT. Cette clause est utilisée pour réduire le nombre de lignes renvoyées par une requête, ce qui peut accélérer l'exécution en réduisant le travail de MySQL.
Voici quelques points clés :

- L'utilisation de LIMIT avec ORDER BY permet à MySQL d'arrêter le tri dès qu'il trouve le nombre requis de lignes, ce qui peut être plus rapide que de trier l'ensemble complet des résultats.
  ```
  SELECT * FROM table ORDER BY colonne DESC LIMIT 10;
  ```
  Cela trie les données par colonne en ordre décroissant et renvoie les 10 premières lignes.
- L'association de LIMIT avec DISTINCT ou GROUP BY peut également améliorer les performances en limitant le travail de calcul nécessaire.
  ```
  SELECT DISTINCT colonne FROM table LIMIT 5;
  ```
  ou
  ```
  SELECT colonne, COUNT(*) FROM table GROUP BY colonne LIMIT 5;
  ```
  Le premier exemple renvoie 5 valeurs distinctes de colonne, tandis que le second exemple retourne les 5 premiers groupes.

L'utilisation de l'instruction *OPTIMIZE TABLE* dans MySQL peut avoir un impact indirect sur l'optimisation des requêtes. En réorganisant la structure physique d'une table et de ses index, cette commande peut réduire l'espace disque nécessaire et améliorer l'efficacité des opérations d'E/S. Pour les tables souvent modifiées, cela peut conduire à une amélioration des performances des requêtes en réduisant le temps d'accès aux données. Cependant, l'effet est plus prononcé pour certaines tables comme MyISAM par rapport à InnoDB, qui gère l'espace plus efficacement.
L'instruction *OPTIMIZE TABLE* peut entraîner un verrouillage de la table durant son exécution. Pour les tables InnoDB, MySQL reconstruit la table pour mettre à jour les statistiques de la table et libérer de l'espace inutilisé. Pour les tables MyISAM, cela réorganise les données et réduit l'espace de fichier. L'utilisation de cette instruction n'est généralement pas nécessaire pour InnoDB sauf dans des cas spécifiques, car InnoDB gère l'allocation de l'espace plus efficacement.

```
OPTIMIZE [NO_WRITE_TO_BINLOG | LOCAL]
    TABLE tbl_name [, tbl_name] …
```

Sortie OPTIMIZE TABLE

OPTIMIZE TABLE renvoie un résultat avec les colonnes indiquées dans le tableau suivant.

Colonne	Valeur
`Table`	Le nom de la table
`Op`	Toujours `optimize`
`Msg_type`	`status`ou `errorinfonotewarning`
`Msg_text`	Un message d'information

L'optimisation des requêtes dans MySQL est un processus dynamique et continu. La compréhension des mécanismes internes de l'optimiseur, la structuration adéquate des requêtes, et la gestion stratégique des index sont cruciales pour maintenir les performances à un niveau optimal. Une surveillance régulière et des ajustements basés sur les tendances de l'utilisation des données peuvent aider à maintenir une base de données performante et efficace.

Bibliographie

Morgan, D. (n.d.). Failover with MySQL utilities part 2: mysqlfailover. Percona Blog. Récupéré de Percona.

MySQL. (n.d.). MySQL 5.7 Reference Manual: MySQL cluster replication and failover. Récupéré de MySQL.

MySQL. (n.d.). MySQL Workbench User Guide. Récupéré de MySQL.

MySQL. (n.d.). MySQL 8.0 Reference Manual: SET TRANSACTION statement. Récupéré de MySQL.

MySQL. (n.d.). MySQL 8.0 Reference Manual: Group replication system variables. Récupéré de MySQL.

MySQL. (n.d.). MySQL Cluster Manager 8.0 User Guide. Récupéré de MySQL.

MySQL. (n.d.). MySQL 8.0 Reference Manual: Security considerations for LOAD DATA LOCAL. Récupéré de MySQL.

MySQL. (n.d.). MySQL 8.0 Reference Manual: Replication asynchronous connection failover. Récupéré de MySQL.

Severalnines. (n.d.). Introduction to failover in MySQL replication. Récupéré de Severalnines.

PHP Manual. (n.d.). mysqlnd_ms_xa_begin. Récupéré de Widgeo.net.

Percona. (n.d.). MySQL 8.0 functional indexes. Récupéré de Percona.

GeeksforGeeks. (n.d.). Architecture of MySQL. Récupéré de GeeksforGeeks.

IBM Cloud. (n.d.). Managing MySQL connections. Récupéré de IBM Cloud.