Administration de MySQL
Configuration et Gestion des droits

Partie 1 : Configuration MySQL

Dans cette section, nous allons apprendre comment configurer un serveur MySQL.
Nous supposons que la partie installation est déjà réalisée.

1. Le fichier de configuration

Le fichier `my.cnf` (ou `my.ini` sous Windows) est le fichier de configuration du serveur MySQL. Les programmes fournis par MySQL (`mysqld`, `mysql`, `mysqldump`...) viennent y chercher leurs directives.

Sous Linux, ils recherchent automatiquement le fichier `my.cnf` dans les répertoires suivants : `/etc/`, `/etc/mysql/` et `~/`. Selon les distributions, d'autres répertoires supplémentaires pourront être utilisés. Par exemple, pour Debian, Ubuntu et leurs dérivés, il est possible d'ajouter des fichiers de configuration supplémentaires dans le répertoire `/etc/mysql/conf.d`. Vous pouvez visualiser la liste des répertoires dans lesquels le serveur cherchera le fichier `my.cnf` avec la commande `mysqladmin --help`. Le résultat est très long, mais vous trouverez une partie à l'allure suivante :

```
MacBook-Pro-de-mohammed:etc mohammedelmalki$
MacBook-Pro-de-mohammed:etc mohammedelmalki$ mysqladmin --help
mysqladmin  Ver 8.0.19 for osx10.14 on x86_64 (Homebrew)
Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Administration program for the mysqld daemon.
Usage: mysqladmin [OPTIONS] command command....
  --bind-address=name IP address to bind to.
  -c, --count=#       Number of iterations to make. This works with -i
                      (--sleep) only.
  -#, --debug[=#]     This is a non-debug version. Catch this and exit.
  --debug-check       This is a non-debug version. Catch this and exit.
  --debug-info        This is a non-debug version. Catch this and exit.
  -f, --force         Don't ask for confirmation on drop database; with
                      multiple commands, continue even if an error occurs.
  -C, --compress      Use compression in server/client protocol.
  --character-sets-dir=name
                      Directory for character set files.
  --default-character-set=name
                      Set the default character set.
  -?, --help          Display this help and exit.
  -h, --host=name     Connect to host.
  -b, --no-beep       Turn off beep on error.
  -p, --password[=name]
                      Password to use when connecting to server. If password is
                      not given it's asked from the tty.
  -P, --port=#        Port number to use for connection or 0 for default to, in
                      order of preference, my.cnf, $MYSQL_TCP_PORT,
                      /etc/services, built-in default (3306).
  --protocol=name     The protocol to use for connection (tcp, socket, pipe,
                      memory).
  -r, --relative      Show difference between current and previous values when
                      used with -i. Currently only works with extended-status.
  -s, --silent        Silently exit if one can't connect to server.
  -S, --socket=name   The socket file to use for connection.
  -i, --sleep=#       Execute commands repeatedly with a sleep between.
  --ssl-mode=name     SSL connection mode.
  --ssl-ca=name       CA file in PEM format.
  --ssl-capath=name   CA directory.
  --ssl-cert=name     X509 cert in PEM format.
  --ssl-cipher=name   SSL cipher to use.
  --ssl-key=name      X509 key in PEM format.
  --ssl-crl=name      Certificate revocation list.
  --ssl-crlpath=name  Certificate revocation list path.
  --tls-version=name  TLS version to use, permitted values are: TLSv1, TLSv1.1,
                      TLSv1.2, TLSv1.3
  --ssl-fips-mode=name
                      SSL FIPS mode (applies only for OpenSSL); permitted
                      values are: OFF, ON, STRICT
  --tls-ciphersuites=name
```

Sous Windows, le principe est similaire. Le serveur va chercher le fichier `my.ini` dans les répertoires suivants : `WINDIR\my.ini`, `WINDIR\my.cnf`, `C:\my.ini`, `C:\my.cnf`, `INSTALLDIR\my.ini` et `INSTALLDIR\my.cnf`, `WINDIR` étant le répertoire Windows (en général `C:\WINDOWS`) et `INSTALLDIR` le répertoire dans lequel MySQL est installé.

Sous Mos, l'ensemble des répertoires se trouvent dans le répertoire : `/usr/local/Cellar/mysqlxxxx`.

Les données sont `/usr/local/var/mysql`.

Le fichier de configuration se trouve : `/usr/local/etc`

Directory	Contents of Directory
bin	**mysqld** server, client and utility programs
data	Log files, databases
docs	Helper documents, like the Release Notes and build information
include	Include (header) files
lib	Libraries
man	Unix manual pages
mysql-test	MySQL test suite
share	Miscellaneous support files, including error messages, sample configuration files, SQL for database installation
support-files	Scripts and sample configuration files
/tmp/mysql.sock	Location of the MySQL Unix socket

Si votre fichier ne se trouve pas dans l'un de ces répertoires ou s'il ne se nomme pas `my.cnf`, vous devrez alors indiquer aux programmes (`mysqld`, `mysql`, `myisamchk`...) son emplacement en le plaçant en paramètre lors de l'appel du programme. Utilisez l'option `defaults-file` qui permet d'indiquer au programme l'emplacement du fichier de configuration. Elle doit toujours être indiquée en première position. L'option `defaults-extra-file` permet de lire un fichier de configuration secondaire après avoir lu le principal.

```
$ mysql --defaults-file=/usr2/mysql/conf/mysql_client.cnf ...

$ mysqld --defaults-file=/usr2/mysql/conf/mysql_server.cnf ...
```

2. Structure du fichier de configuration

Le fichier de configuration est organisé en sections (ou groupes). Une section est composée d'un nom (en général, le nom du programme référencé) placé entre deux crochets, d'options, et elle se termine par le début d'une autre section ou la fin du fichier. Ainsi, le nom de la section du serveur MySQL est `[mysqld]`. Toutefois, il est possible d'utiliser `[server]`. Le nom de la section du client en texte par défaut est `[mysql]` (à ne pas confondre avec `[mysqld]`). Suivant le même principe, vous pouvez utiliser la section `[mysqldump]` qui permet de paramétrer le client texte permettant de faire des sauvegardes `mysqldump`. Il existe aussi la section `[client]` qui permet de mutualiser les informations de configuration de tous les clients (`mysql`, `mysqldump`, `mysqladmin`...).

La syntaxe des options est de la forme nom_option=valeur. Les options composées de plusieurs mots peuvent indifféremment être séparées par un tiret « - » ou un tiret bas « _ ». Certaines options sont binaires, la fonctionnalité est alors activée simplement en inscrivant le nom de l'option. C'est le cas par exemple de l'option `log-slave-updates`. Il est en outre possible de mettre des commentaires dans le fichier de configuration, il suffit de préfixer les lignes par le caractère dièse « # » ou un point-virgule « ; ».

Vérifiez bien la syntaxe des options écrites dans votre fichier de configuration, car le serveur refusera de démarrer s'il ne comprend pas une option de la section `[mysqld]`. Il générera une erreur qui sera écrite dans le journal des erreurs.

```
$ cat /etc/mysql/my.cnf

# Section globale à tous les clients MySQL
[client]
socket = /tmp/mysql.sock # les clients se connectent avec ce socket

# Section du client texte mysql
[mysql]
prompt = \R:\m \u$\d> # L'invite de commande du client texte mysql
est de la forme : HH:mm user$schema>

# Section du serveur MySQL
[mysqld]
```

```
slow_query_log # activation du journal des requêtes lentes
slow_query_log_file = mysql-slow.log # nom du journal des requêtes
lentes
long-query-time = 2 # Les requêtes dont la durée est supérieure à
2 secondes sont journalisées
enable-federated # Active le moteur FEDERATED
```

La configuration du serveur peut être éclatée dans plusieurs fichiers. L'instruction `!include` permet d'inclure le contenu d'un fichier dans un autre et l'instruction `!includedir` permet d'analyser un répertoire dans lequel le ou les fichiers de configuration qui ont pour extension `.cnf` sous Linux (`.ini` et `.cnf` sous Windows) seront lus. Vous pouvez par exemple vous en servir pour créer un fichier par groupes fonctionnels (`replication.cnf`, `log.cnf`, `innodb.cnf`...). Attention toutefois, vous n'avez pas la possibilité de garantir l'ordre de lecture des fichiers avec `!includedir`.

Lorsqu'on utilise des fichiers de configuration supplémentaires grâce à l'option `!includedir`, une erreur courante est de ne pas spécifier une extension `.cnf` aux fichiers supplémentaires. Les fichiers seront alors simplement ignorés, sans aucun message d'erreur ou d'avertissement.

3. Paramétrage dynamique du serveur

Redémarrer MySQL pour prendre en compte des modifications de configuration n'est pas toujours très pratique. Heureusement, de plus en plus d'options peuvent être changées dynamiquement (donc sans redémarrage du serveur) grâce à la commande `SET` à condition toutefois que l'utilisateur possède le droit `SUPER`. Le changement peut se faire de deux manières, soit au niveau local à la connexion, c'est-à-dire seulement pour la session cliente en cours grâce à la clause `SESSION` ou alors de façon globale au serveur, à l'aide de la clause `GLOBAL`. La plupart des options ont une portée session et globale (comme l'option `sql_mode` qui est vue plus loin dans ce chapitre), mais ce n'est pas le cas de toutes.

a. Changement pour la session

En utilisant la commande `SET SESSION`paramètre_session=valeur, la valeur du paramètre va alors changer seulement pour la session du client dans lequel la commande a été effectuée.
Ici, on voit que la variable a la même valeur globale et pour la session :

```
mysql> SHOW GLOBAL VARIABLES LIKE 'sort_buffer_size';
+------------------+--------+
| Variable_name    | Value  |
+------------------+--------+
| sort_buffer_size | 262144 |
+------------------+--------+
1 row in set (0,01 sec)

mysql> SHOW session VARIABLES LIKE 'sort_buffer_size';
+------------------+--------+
| Variable_name    | Value  |
+------------------+--------+
| sort_buffer_size | 262144 |
+------------------+--------+
1 row in set (0,05 sec)

mysql>
```

⇒ Changer la valeur par défaut.
⇒ Vérifier que la nouvelle valeur a bien été prise en charge

b. Changement global

Avec la commande `SET GLOBAL` paramètre_global=valeur, le changement a lieu pour tous les clients qui se connectent au serveur après l'exécution de la commande. Cependant cette nouvelle valeur n'est pas prise en compte pour les sessions des clients connectés avant le changement (sessions en cours), ils gardent donc la valeur initiale du paramètre. C'est également le cas pour la session du client dans lequel la commande a été effectuée.

Une exception à cette règle concerne les paramètres qui n'ont qu'une portée globale. La mise à jour a lieu également dans la session du client qui effectue la commande `SET GLOBAL`.

⇒ Changer la valeur par défaut.
⇒ Vérifier que la nouvelle valeur a bien été prise en charge
⇒ Vérifier la valeur de la session, que constatez-vous ?

Il est très important de garder à l'esprit que ces changements à chaud ne sont pas persistants. Ils sont stockés dans l'espace mémoire utilisé par le serveur MySQL et ne survivent donc pas à un redémarrage du serveur. Les changements effectués avec la commande `SET SESSION` ne survivent pas également à un redémarrage du client car cela initie une nouvelle session. Si vous voulez rendre vos changements globaux persistants, vous devez les mettre dans le fichier de configuration.

⇒ Redémarrez votre session.
⇒ Reverfier les valeurs session et global
⇒ Procédez à la modification de ces dernières mais en les persistants cette fois-ci.

MySQL 8.0 a introduit une nouvelle syntaxe pour changer dynamiquement une option tout en enregistrant la nouvelle valeur dans un fichier de configuration spécial. La syntaxe utilise `SET PERSIST` et elle est très simple à comprendre sur un exemple :

On vérifie d'abord la valeur de l'option `sort_buffer_size` :

On vérifie d'abord la valeur de l'option `sort_buffer_size` :

```
mysql> SHOW GLOBAL VARIABLES LIKE 'sort_buffer_size';
+------------------+--------+
| Variable_name    | Value  |
+------------------+--------+
| sort_buffer_size | 262144 |
+------------------+--------+
```

Puis on modifie cette option en enregistrant les modifications :

```
mysql> SET PERSIST sort_buffer_size = 5242880;
```

On vérifie que la modification est prise en compte :

```
mysql> SHOW GLOBAL VARIABLES LIKE 'sort_buffer_size';
+------------------+---------+
```

```
| Variable_name    | Value   |
+------------------+---------+
| sort_buffer_size | 5242880 |
+------------------+---------+
```

On peut ensuite vérifier qu'un nouveau fichier (`mysqld-auto.cnf`) a été créé dans le répertoire de données. Ce fichier sera lu si MySQL est redémarré :

```
# cat /var/lib/mysql/mysqld-auto.cnf
{
   "Version" : 1 ,
   "mysql_server" : {
      "sort_buffer_size" : {
         "Value" : "5242880" ,
         "Metadata" : {
            "Timestamp" : 1530191568276345 ,
            "User" : "root" ,
            "Host" : "localhost"
         }
      }
   }
}
```

Attention, même si `SET PERSIST` peut être pratique, il peut être dangereux dans le cas où le déploiement de la configuration est automatisé par des outils tels que Chef, Puppet ou Ansible. Dans ce cas, les modifications apportées par `SET PERSIST` ne seront pas prises en compte par le système de déploiement et les instances de MySQL créées ensuite ne bénéficieront pas des changements enregistrés avec `SET PERSIST`. Dans ce cas, il est conseillé de mettre la variable `persisted_globals_load` à `OFF`, ce qui désactive le chargement du fichier `mysqld-auto.cnf` au démarrage de MySQL

4. Visualisation de la configuration

Regarder le contenu du fichier `my.cnf` permet de connaître la valeur de la majorité des options, mais cette technique est loin d'être fiable. En effet, certaines options sont peut-être définies dans des fichiers de configuration supplémentaires, ou peut-être qu'un utilisateur a modifié dynamiquement un paramètre sans faire le changement correspondant dans le fichier `my.cnf`. Par conséquent, il est préférable de se connecter au serveur et d'exécuter l'une des requêtes suivantes (ici pour connaître la valeur de `sort_buffer_size`) :

```
mysql> SHOW GLOBAL VARIABLES LIKE 'sort_buffer_size';
+------------------+---------+
| Variable_name    | Value   |
+------------------+---------+
| sort_buffer_size | 5242880 |
+------------------+---------+
mysql> SELECT @@global.sort_buffer_size;
+-----------------------------+
| @@global.sort_buffer_size   |
+-----------------------------+
|                     5242880 |
+-----------------------------+
```

⇒ Vous pouvez alors comparer avec le fichier `my.cnf`

⇒ Ajuster le fichier de configuration si besoin.

- Pour la valeur d'une variable de session, utilisez `SHOW SESSION VARIABLES` ou `SELECT @@SESSION.`

```
mysql> SELECT * FROM performance_schema.variables_info WHERE
VARIABLE_NAME='innodb_buffer_pool_size';
+--------------------------+----------------+---------------+-----------+---------------------
+----------+----------+----------+
| VARIABLE_NAME            | VARIABLE_SOURCE | VARIABLE_PATH | MIN_VALUE | MAX_VALUE
| SET_TIME | SET_USER | SET_HOST |
+--------------------------+----------------+---------------+-----------+---------------------
+----------+----------+----------+
| innodb_buffer_pool_size | COMPILED        |               | 5242880   | 9223372036854775807
| NULL     | NULL     | NULL     |
+--------------------------+----------------+---------------+-----------+---------------------
+----------+----------+----------+
1 row in set (0,00 sec)
```

Une valeur par défaut :

```
mysql> SELECT * FROM performance_schema.variables_info WHERE
VARIABLE_NAME='innodb_buffer_pool_size'\G
*************************** 1. row ***************************
 VARIABLE_NAME: innodb_buffer_pool_size
VARIABLE_SOURCE: COMPILED
[...]
```

Une variable modifiée dynamiquement :

```
mysql> SELECT * FROM performance_schema.variables_info WHERE
VARIABLE_NAME='sort_buffer_size'\G
*************************** 1. row ***************************
 VARIABLE_NAME: sort_buffer_size
VARIABLE_SOURCE: DYNAMIC
[...]
```

Une variable dont la valeur provient d'un fichier de configuration :

```
mysql> SELECT * FROM performance_schema.variables_info WHERE
VARIABLE_NAME='socket'\G
*************************** 1. row ***************************
 VARIABLE_NAME: socket
VARIABLE_SOURCE: GLOBAL
 VARIABLE_PATH: /etc/mysql/my.cnf
[...]
```

Il est alors assez simple d'écrire des requêtes permettant de déterminer toutes les options modifiées dynamiquement ou toutes les options modifiées dans un fichier de configuration en particulier.

Notez que les requêtes précédentes ne donnent malheureusement pas la valeur des options. Il vous faudra donc toujours avoir recours à `SHOW GLOBAL/SESSION VARIABLES` pour connaître la valeur des paramètres.

5. Autres paramètres à configurer

a) Nombre de connexions simultanées

MySQL utilise la variable `max_connections` pour limiter le nombre de connexions simultanées au serveur. Deux indicateurs peuvent vous alerter quand cette valeur est trop faible : la variable de

statut `Max_used_connections` vaut ou est proche de `max_connections` ou le serveur refuse des connexions avec l'erreur `Too many connections`.

Néanmoins, mieux vaut bien réfléchir aux conséquences possibles d'une augmentation de la valeur de ce paramètre, car son but est d'empêcher le serveur de s'écrouler sous une charge trop importante. Si la valeur de `max_connections` nécessaire pour éviter l'erreur `Too many connections` s'avère tellement élevée que les performances de tout le serveur se dégradent, il faut sans doute penser à d'autres modifications (réplication, changement de matériel, changements applicatifs...).

⇒ Modifiez le nombre de connexion par défaut (4 par exemple)
⇒ Ouvrez 4 sessions mysql
⇒ Ouvre une cinquième session, que se passe-t-il ?

MySQL conserve toujours une connexion disponible en local pour l'utilisateur `root`, en cas de problème. C'est pourquoi `max_connections` vaut par défaut 151 et non 150, autorisant effectivement 150 connexions applicatives.

b) Caches de table

Ces caches stockent des informations sur les tables, ce qui permet, quand le serveur a besoin d'utiliser une table, d'en accélérer l'accès.

Le cache est divisé en deux parties : un cache des tables ouvertes (`table_open_cache`) et un cache de définition de table (`table_definition_cache`). L'ajustement de ces deux paramètres se fait comme en utilisant les variables de statut suivantes :

- `Opened_tables` pour `table_open_cache`. Si la valeur de cette variable augmente régulièrement, il est recommandé d'augmenter la valeur de `table_open_cache`.
- `Opened_table_definitions` pour `table_definition_cache`, sur le même principe. En général, le plus simple est de configurer `table_definition_cache` à une valeur légèrement plus élevée que le nombre de tables existant sur le serveur.

```
mysql> show global status  like 'open%';
  +---------------------------+-------+
  | Variable_name             | Value |
  +---------------------------+-------+
  | Open_files                | 2     |
  | Open_streams              | 0     |
  | Open_table_definitions    | 45    |
  | Open_tables               | 83    |
  | Opened_files              | 2     |
  | Opened_table_definitions  | 71    |
  | Opened_tables             | 161   |
  +---------------------------+-------+
        7 rows in set (0,01 sec)
```

Si vous avez plusieurs dizaines de milliers de tables dans votre système, il n'est en général pas conseillé d'utiliser des valeurs au-delà de 10000 pour ces deux caches car des problèmes de performances peuvent alors apparaître. Il existe également la variable `table_open_cache_instances`, qui permet d'utiliser plusieurs instances du `table_open_cache`. Des valeurs supérieures à 10000 pour `table_open_cache` sont ainsi rendues possibles avec cette option.

⇒ Modifier légèrement les valeurs de cache pour les tables

c) Cache de threads

Quand une connexion au serveur MySQL se termine, le thread associé à cette connexion est placé dans un cache afin de pouvoir être réutilisé, évitant une opération de création d'un nouveau thread.

La variable correspondante est `thread_cache_size`, et pour l'ajuster il suffit de regarder l'évolution de la variable de statut `Threads_created` au cours du temps : en augmentant la taille du cache, vous pourrez souvent parvenir à faire en sorte que la valeur de `Threads_created` augmente de moins de 5 par seconde.

⇒ Changez la valeur par défaut
⇒ Visualiser cette valeur
⇒ La valeur affectée a-t-elle été enregistré dans le fichier de configuration ?

La création d'un thread est rapide avec MySQL (encore plus avec MySQL 5.7), il ne faut donc pas espérer des gains très importants en augmentant la taille du cache de threads.

d) Paramètres MyISAM

Le seul réel paramètre important pour les tables `MyISAM` est la taille du cache d'index, commandée par la variable `key_buffer_size`.

⇒ La valeur par défaut est-elle correctement définie ?

Pour vous aider à trouver une bonne taille, vous allez regarder les valeurs de trois variables de statut parmi le résultat de la commande :

```
mysql> SHOW GLOBAL STATUS LIKE 'Key_%';
```

Les deux premières variables à considérer sont `Key_reads` et `Key_read_requests`. `Key_reads` indique le nombre de requêtes de lectures d'index qui n'ont pas pu être satisfaites par le cache et `Key_read_requests` indique le nombre de lectures faites dans l'index. Si vous calculez le taux d'utilisation du cache par la formule `(1 - Key_reads / Key_read_requests) x 100`, votre cache devrait être bien paramétré si votre taux est proche de 100 %. Si le taux est faible, votre cache est sans doute trop petit et vous pouvez augmenter la valeur de `key_buffer_size`.

La troisième variable à regarder est `Key_blocks_unused`, qui vous donne le nombre de blocs disponibles dans le cache. Si votre taux d'utilisation est faible et que le nombre de blocs disponibles est également faible, votre cache est très certainement trop petit.

Sachez qu'il n'est pas pénalisant de prévoir un cache surdimensionné, car la mémoire ne sera allouée qu'en cas de besoin, contrairement au cache de requêtes.

Il n'existe pas d'option pour mettre en cache les données des tables `MyISAM`. Seul le système d'exploitation met en cache les données, ce qui est cependant beaucoup moins efficace qu'un cache spécialisé comme celui existant pour les index.

Paramètres à ne pas modifier

Cette section peut paraître étrange, mais quelques paramètres ont une valeur par défaut qui est très faible et de nombreuses personnes essaient de configurer une valeur plus élevée dans l'espoir d'améliorer les performances. Malheureusement, au mieux, l'impact sera invisible, et dans le pire des cas, vous pourrez constater des dégradations de performances.

Ne modifiez donc jamais (sauf si vous comprenez parfaitement l'impact de vos changements) `read_buffer_size`, `read_rnd_buffer_size`, `join_buffer_size` et `sort_buffer_size`.

6. Configuration d'InnoDB

a. Paramètres essentiels

Les paramètres évoqués dans cette section ne sont pas nécessairement à modifier par rapport à la configuration par défaut, mais il est important que vous ayez réfléchi à leur bonne valeur pour votre application.

- Taille du cache mémoire (`innodb_buffer_pool_size`) : il s'agit du cache principal d'`InnoDB` (buffer pool), où données et index fréquemment accédés sont stockés. Pour un serveur dédié à MySQL, il est courant de lui allouer la majeure partie de la mémoire du serveur (par exemple, environ 25 Go pour un serveur ayant 32 Go de mémoire physique). L'idée principale est que ce cache permet d'éviter les accès au disque : la taille du cache est d'autant plus importante que le disque est lent.

  Si votre base de données est petite (quelques dizaines de Go, par exemple), il est assez simple d'utiliser un serveur ayant suffisamment de mémoire pour que l'ensemble des données et index `InnoDB` tiennent dans le cache. Sinon, il faut essayer de faire tenir en cache la partie utile des données et index, c'est-à-dire la partie des données et index qui est fréquemment utilisée par l'application. Il est assez fréquent en effet d'avoir par exemple une base de 500 Go contenant l'historique des données sur les cinq dernières années, mais pour laquelle seules les données du dernier mois sont régulièrement consultées, ce qui va représenter 10 Go. Dans ce cas, il est inutile de chercher à allouer 500 Go au buffer pool, 10-15 Go seront suffisants.
- Taille du journal transactionnel (`innodb_log_file_size`) : le journal transactionnel (*redo log*) permet à `InnoDB` d'offrir de bonnes performances en écriture tout en garantissant l'intégrité des données en cas d'arrêt inopiné (voir chapitre Généralités sur MySQL). Pour rappel, l'idée principale est qu'`InnoDB` écrit de manière synchrone les modifications dans son journal transactionnel (écritures peu coûteuses puisque séquentielles) et qu'un thread en arrière-plan vient modifier les fichiers de données de manière asynchrone (écritures très coûteuses car aléatoires). Cependant, la taille du journal transactionnel est limitée, donc si vous écrivez plus vite dans le journal transactionnel que la purge d'arrière-plan fait de la place dans le journal transactionnel, `InnoDB` risque de bloquer toutes les écritures pendant quelques secondes le temps que le processus d'arrière-plan récupère de l'espace libre dans le journal transactionnel. Ce comportement est catastrophique pour les performances en écriture, il est par conséquent particulièrement important de dimensionner correctement le journal transactionnel. La valeur par défaut de 48 Mo conviendra aux applications qui écrivent peu mais sera très insuffisante si le volume d'écriture est plus élevé.

  Essayez de dimensionner le journal transactionnel pour qu'il puisse contenir environ 1 heure de modifications en heure de pointe. Cela se fait simplement en comparant à 1 heure d'intervalle la valeur de la variable `Log sequence number` qui apparaît dans la section `LOG` de la sortie de `SHOW ENGINE INNODB STATUS`. Des valeurs telles que 256/512/1024 Mo sont courantes.
- Durabilité (`innodb_flush_log_at_trx_commit`) : les explications concernant ce paramètre se trouvent à la section suivante.
- Tablespace pour chaque table (`innodb_file_per_table`) : dans les anciennes versions de MySQL, toutes les données et index de toutes les tables étaient stockées dans le tablespace partagé (`ibdata1`). Cela ne posait aucun problème de performances mais de nombreux administrateurs n'appréciaient pas particulièrement de

se retrouver avec un seul énorme fichier contenant toutes les données. Ce fichier pouvait dépasser le To si la base était suffisamment volumineuse. Pour éviter ce désagrément, la solution a toujours été d'utiliser `innodb_file_per_table = 1`, ce qui indique à `InnoDB` de stocker chaque table dans son propre fichier `.ibd`. Il s'agit de la valeur par défaut dorénavant et son principal avantage est de pouvoir récupérer l'espace disque des tables supprimées. Notez cependant qu'`innodb_file_per_table` ne procure aucun avantage en performances. Enfin, notez qu'il existe un cas où il est conseillé de ne pas utiliser cette option : si vous avez de nombreuses tables dans votre schéma, par exemple plusieurs dizaines de milliers. Dans ce cas, la plupart des tables sont en général petites, et `innodb_file_per_table` gaspille de l'espace disque pour chaque table, ce qui finit par devenir gênant quand le nombre de tables est très élevé.

- Méthode de flush des données (`innodb_flush_method`) : par défaut, quand des modifications sont écrites dans les fichiers de données, une partie au moins est également conservée dans le cache du système de fichiers. En général, cette mise en cache est totalement inutile car `InnoDB` dispose de son propre cache (le buffer pool). Bien souvent, si vos disques sont rapides, vous préférerez éviter cette double mise en cache en spécifiant `innodb_flush_method = O_DIRECT`. Attention, si vos disques sont lents, `O_DIRECT` peut avoir un réel impact négatif sur les performances de tous les accès disque, puisque le cache du système de fichiers est simplement ignoré.

Il est également intéressant de noter que MySQL 8.0 a introduit une nouvelle option appelée `innodb_dedicated_server`. Si vous activez cette option, ce qui nécessite un redémarrage de MySQL, le serveur choisira automatiquement les valeurs des options `innodb_buffer_pool_size`, `innodb_log_file_size` et `innodb_flush_method`. Attention, `innodb_dedicated_server` suppose que MySQL peut utiliser toutes les ressources de la machine, veillez donc à n'activer cette option que sur un serveur dédié à MySQL. Cette option n'intéressera sans doute pas les DBA expérimentés, qui préféreront pouvoir choisir individuellement les valeurs des différents paramètres, mais il s'agit là d'un moyen très simple de s'assurer que la configuration de base d'`InnoDB` est correcte.

b. Isolation et durabilité

Isolation et durabilité sont le I et le D de l'acronyme ACID qui a été évoqué au chapitre Généralités sur MySQL. Comprendre ces deux notions est important pour un bon paramétrage d'`InnoDB` : il est en effet possible d'influer significativement sur les performances du moteur en faisant varier le niveau d'isolation ou le niveau de durabilité. Mais l'impact sur la sécurité des données est lui aussi significatif, prenez donc le temps de bien comprendre les avantages et inconvénients de chaque type de paramétrage.

i. Réglage de l'isolation

Dans l'acronyme ACID qui décrit les propriétés vérifiées par les systèmes transactionnels, le I signifie isolation, c'est-à-dire que les opérations réalisées dans une transaction n'ont pas d'effet sur les autres transactions en cours. Il est cependant possible de paramétrer le serveur pour que cette isolation soit plus ou moins réalisée.

Selon le niveau d'isolation, trois problèmes peuvent survenir dans les transactions :

- Lecture sale (*dirty read*) : ce problème intervient quand une transaction peut lire des lignes modifiées par une autre transaction mais pas encore validées par un `COMMIT`.
- Lecture non répétable (*non-repeatable read*) : ce problème intervient quand on exécute plusieurs fois une même requête `SELECT` et que le serveur ne renvoie pas le même résultat, parce que d'autres transactions ont modifié ces lignes et les ont validées par un `COMMIT`.
- Lecture fantôme (*phantom read*) : ce problème intervient quand des données insérées par d'autres transactions deviennent visibles.

Il existe quatre niveaux d'isolation, définis dans le standard SQL et supportés par `InnoDB` :

- `READ UNCOMMITTED` : les changements faits par d'autres transactions et non validés par un `COMMIT` sont visibles. Les lectures sales, les lectures non répétables et les lectures fantômes sont possibles.

- READ COMMITTED : les changements faits par d'autres transactions et validés par un COMMIT sont visibles. Les lectures non répétables et les lectures fantômes sont possibles.
- REPEATABLE READ : dans ce niveau d'isolation, si une même requête est exécutée plusieurs fois dans une même transaction, le serveur renverra toujours le même résultat. Seules les lectures fantômes sont possibles. C'est le niveau d'isolation par défaut d'InnoDB.
- SERIALIZABLE : chaque transaction est complètement isolée des autres transactions. Aucun des trois problèmes cités n'est alors possible.

Le niveau d'isolation SNAPSHOT d'Oracle et SQL Server n'existe pas avec MySQL.

InnoDB maintient l'isolation entre les transactions en conservant autant que nécessaire les anciennes versions des lignes de manière à ce que plusieurs transactions différentes puissent voir chacune la version correcte des données. Maintenir plusieurs versions des données sur de nombreuses lignes a cependant un coût.

Le niveau d'isolation se modifie en utilisant la variable transaction-isolation. Les valeurs possibles sont READ-UNCOMMITTED, READ-COMMITTED, REPEATABLE-READ et SERIALIZABLE.

Sachez qu'en général, le niveau par défaut est le meilleur compromis entre performances et bonne isolation des transactions. Cependant, il peut être très intéressant d'utiliser le niveau READ COMMITTED si le serveur doit régulièrement exécuter des requêtes très lentes (comme des requêtes de reporting) : dans ce cas, READ COMMITTED permet de maintenir pendant moins longtemps les anciennes versions des lignes, ce qui peut avoir un impact positif significatif sur les performances.

ii. Réglage de la durabilité

InnoDB enregistre les informations de chaque transaction dans un cache afin d'éviter de solliciter le disque. Ces données résidant en mémoire, elles ne résistent pas à un crash MySQL et encore moins à un crash du serveur entier. Au moment du COMMIT, ces informations sont recopiées dans les journaux d'InnoDB pour pouvoir être ensuite appliquées sur les fichiers de données (tables et index) concernés.

Dès que les données sont copiées dans un journal, elles peuvent être considérées comme étant en sécurité. En effet, même en cas d'arrêt inopiné, InnoDB peut relire le journal pour opérer les modifications demandées sur les tables. Le mécanisme mis en place par InnoDB permet donc bien d'assurer la durabilité des changements exécutés lors d'une transaction (le D de l'acronyme ACID).

Il existe une difficulté supplémentaire au moment de l'écriture sur le fichier de log : le système d'exploitation peut décider d'écrire non pas sur le disque, mais dans son cache. Dans les deux cas, InnoDB aura l'impression que l'écriture a eu lieu sur le disque, mais si les données sont dans le cache du système d'exploitation, elles sont en réalité en mémoire et seront donc perdues en cas d'arrêt inopiné. C'est pourquoi InnoDB demande au système d'exploitation de faire un flush après l'écriture pour s'assurer que les données vont bien se retrouver sur le disque. Cette opération est coûteuse en termes de performances.

InnoDB vous laisse le contrôle sur la qualité de la durabilité qui est mise en œuvre avec l'option innodb_flush_log_at_trx_commit. Les trois valeurs possibles sont 0, 1 ou 2 :

- 1 est la valeur par défaut. Cela signifie qu'InnoDB transfère les informations du cache vers le journal et fait un flush du journal à chaque COMMIT.
- 2 indique à InnoDB de transférer les informations du cache vers le journal à chaque COMMIT, mais ne fait un flush du journal qu'environ une fois par seconde.
- 0 indique à InnoDB de transférer les informations du cache vers le journal et de faire un flush du journal environ une fois toutes les secondes.

Il reste à expliquer la différence entre les valeurs 0 et 2, qui peut ne pas sembler évidente.

Avec la valeur 2, à chaque `COMMIT`, les informations concernant la transaction sont, au pire, dans le cache du système d'exploitation et, au mieux, sur le disque. Cela signifie que si vous subissez un crash MySQL (mais pas du serveur entier) entre le moment du `COMMIT` et le moment où le fichier de log est écrit sur le disque (ce qui intervient au maximum une seconde plus tard), les informations de la transaction résident dans le cache du système d'exploitation et peuvent être récupérées. Au contraire, avec la valeur 0, les informations de la transaction n'existent que dans le cache et sont par conséquent perdues en cas de crash MySQL.

Par contre, en cas de crash de la machine hôte, vous risquez de perdre au plus une seconde de données avec les valeurs 0 ou 2. Seule la valeur 1 vous assure de ne pas perdre de données.

En résumé, quand vous avez besoin d'assurer la durabilité au sens ACID du terme, la valeur 1 est impérative. Sinon, la valeur 2 est sans doute un bon compromis entre performances et sécurité des données.

7. Autres paramètres

Voici une liste de paramètres qu'il peut être utile de modifier :

- `innodb_buffer_pool_instances` : si votre buffer pool est très gros (plusieurs dizaines ou centaines de Go), il est possible que certaines opérations subissent des contentions. Il est alors conseillé d'utiliser plusieurs instances de buffer pool pour diminuer les contentions. La valeur par défaut de 8 est souvent correcte.

  ⇒ Vérifiez la valeur par défaut

- `innodb_buffer_pool_dump_at_shutdown`, `innodb_buffer_pool_load_at_startup` : ces deux options permettent de prendre une empreinte du buffer pool avant d'arrêter le serveur et de recharger le buffer pool au démarrage. L'intérêt est de plus ou moins préserver le buffer pool, qui est l'élément le plus critique d'`InnoDB` pour la performance. Ces options sont activées par défaut.

  ⇒ Vérifiez les valeurs par défaut

- `innodb_io_capacity`, `innodb_io_capacity_max` :les processus en arrière-plan d'`InnoDB` limitent leur activité disque afin de ne pas pénaliser les activités d'avant-plan. Cependant, si vous avez des disques rapides et beaucoup d'écritures, vous risquez de constater que les processus d'arrière-plan n'utilisent pas toutes les capacités des disques. C'est le signe qu'il faut augmenter la valeur de ces deux variables. Choisir une bonne valeur est assez difficile puisqu'une valeur trop élevée va inciter `InnoDB` à plus utiliser le disque pour les processus d'arrière-plan, ce qui peut également provoquer des problèmes de performances. L'unité pour ces deux options étant le nombre d'opérations sur disque par seconde, il vous faut donc savoir combien d'opérations votre disque peut réaliser pour configurer ce paramètre. Par exemple, si votre disque peut réaliser 2000 opérations par seconde, `innodb_io_capacity = 1000` et `innodb_io_capacity_max = 1800` pourraient être de bonnes valeurs.

Enfin, si vos données sont stockées sur des disques SSD, vous pouvez ajuster les paramètres suivants :

- `innodb_flush_neighbors = 0` : par défaut (valeur de l'option : `1`) quand une page du buffer pool est écrite sur le disque, `InnoDB` essaie de détecter si une ou plusieurs pages voisines pourraient aussi être écrites au même moment. L'idée est de minimiser le nombre d'écritures aléatoires sur le disque, qui sont traditionnellement très coûteuses. Cette optimisation n'a pas lieu d'être avec des disques SSD, c'est pourquoi la valeur `0` est recommandée.
- `innodb_read_io_threads = 16`, `innodb_write_io_threads = 16` : les disques SSD fonctionnent au mieux lorsque plusieurs opérations ont lieu simultanément, contrairement aux disques classiques.

Partie 2 : Gestion des droits

Objectif :
Définir une politique de sécurité pour les utilisateurs :

On se contraint à utiliser des utilisateurs sans mention de l'hôte (i.e. l'hôte est %). On propose de créer les utilisateurs suivants avec les privilèges correspondants :

- PAS d'utilisateur anonyme :
  - Révoquer tous les droits des utilisateurs anonymes (s'ils existent)
  - Détruire l'utilisateur anonyme ''@% (s'il existe)
- Utilisateur « interface de gestion de la base» :
  - Nom : admin_interface
  - C'est l'utilisateur backoffice en lien avec le gestionnaire de la base
  - Privilèges uniquement liés à la BD.
  - Consultation, modification, ajout, suppression pour toutes les tables
  - Exécution des procédures et fonctions
  - Visualisation des vues
  - Ne peut pas déléguer ses droits à un autre utilisateur (ce n'est pas un utilisateur humain)
- Utilisateur « cust_interface »
  - C'est l'utilisateur frontoffice en lien avec le client.
  - Privilèges uniquement liés à la BD
  - Droits de base
    - Droit de visualiser tout le contenu des tables : deux tables de votre choix.
      GRANT SELECT sur ces tables
    - Droit d'insérer et de modifier des données de client dans la table de votre choix.
      GRANT INSERT et GRANT UPDATE sur cette table

QUESTION : Donner les requêtes permettant de tester les privilèges des deux utilisateurs (voir la liste des requêtes ci-dessous) :

- Une requête permettant de montrer que admin_interface a le privilège SELECT sur une des tables au niveau de laquelle vous avez mis des restrictions à la question precedente.
- Une requête permettant de montrer que admin_interface n'a pas le privilège CREATE TABLE dans la base.
- Une requête permettant de montrer que admin_interface n'a pas le privilège CREATE DATABASE.
- Une requête permettant de montrer que cust_interface a le privilège SELECT sur une des tables au niveau de laquelle vous

avez mis des restrictions à la question précédente.

- Une requête permettant de montrer que cust_interface n'a pas le privilège SELECT sur une des tables au niveau de laquelle vous avez mis des restrictions à la question précédente
- Une requête permettant de montrer que cust_interface a le privilège INSERT sur latable sur une des tables au niveau de laquelle vous avez mis des restrictions à la question précédente
- Une requête permettant de montrer que cust_interface n'a pas le privilège CREATE TABLE dans la base
- Une requête permettant de montrer que cust_interface n'a pas le privilège CREATE DATABASE