MySQL - Moteurs de table

Moteur de table et types de table

On peut créer une table en spécifiant la clause ENGINE ou TYPE :
CREATE TABLE t (i INT) ENGINE = INNODB ;
CREATE TABLE t (i INT) TYPE = MEMORY ;
si on omet cette clause, le moteur par défaut est utilisé, spécifié par la variable système table_type. Pour convertir une table d'un type à l'autre, utilisez la commande ALTER TABLE, pour indiquer le nouveau type :
ALTER TABLE t ENGINE = MYISAM ;
ALTER TABLE t TYPE = BDB ;

Moteur de table MyISAM

Chaque table **MyISAM** est stockée en trois fichiers. Les fichiers portent le nom de la table, et ont une extension qui spécifie le type de fichier. Le fichier **.frm** stocke la définition de la table. L'index est stocké dans un fichier avec l'extension **.MYI** (MYIndex), et les données sont stockées dans un fichier avec l'extension **.MYD** (MYData).

- Toutes les clés numériques sont stockées avec l'octet de poids fort en premier, pour améliorer la compression.
- Support des grands fichiers
- Les lignes de taille dynamique sont bien moins fragmentées lors de l'utilisation d'insertion et d'effacement.
- Le nombre maximal d'index par table est de 64
- La taille maximale d'une clé est de 1000 octets
- Les colonnes **BLOB** et **TEXT** peuvent être indexées.
- Les valeurs NULL sont autorisées dans une colonne indexée. Elles prennent 0 à 1 octets par clé.
- Les fichiers d'index sont généralement plus petits en MyISAM qu'en ISAM.
- améliore l'utilisation d'espace dans l'arbre des clés.
- Vous pouvez placer les fichiers de données et d'index dans différents dossiers pour obtenir plus de vitesse avec les options de table **DATA DIRECTORY** et **INDEX DIRECTORY**, dans la commande **CREATE TABLE**.
- chaque colonne de caractères peut avoir un jeu de caractères distinct.
- Il y a un indicateur dans le fichier **MyISAM** qui indique si la table a été correctement fermée.
- myisampack peut compresser des colonnes **BLOB** et **VARCHAR**.
- Support du vrai type **VARCHAR** ; une colonne **VARCHAR** commence avec une taille, stockée sur 2 octets.
- Les tables ayant des colonnes **VARCHAR** peuvent avoir un format de lignes fixe ou dynamique.
- **VARCHAR** et **CHAR** peuvent prendre jusqu'à 64 ko.
- Un index de hashage peut être utilisé avec **UNIQUE**. Cela vous permettra d'avoir un index **UNIQUE** sur toute combinaison de colonnes de la table. Vous ne pourrez pas utiliser un index **UNIQUE** pour une recherche.

Les options suivantes de mysqld permettent de modifier le comportement des tables MyISAM :

–myisam-recover=mode Active le mode de restauration automatique des tables MyISAM corrompues.

–delay-key-write=ALL N'écrit pas les buffers de clés entre deux écritures dans une table MyISAM.

Note : Si vous faîtes cela, vous ne devez pas utiliser les tables MyISAM avec d'autres programmes (comme depuis un autre serveur MySQL ou avec myisamchk) lorsque la table est utilisée. Sinon, vous allez obtenir une corruption d'index.

Utiliser **–external-locking** n'aidera pas les tables qui utilisent **–delay-key-write**.

Les variables systèmes suivantes affectent le comportement des tables MyISAM

bulk_insert_buffer_size La taille du cache d'index lors des insertions de masse. Note : c'est une limite par thread !

myisam_max_extra_sort_file_size Utilisée pour aider MySQL à décider quand utiliser le cache de clé lent mais sûr.

myisam_max_sort_file_size N'utilise pas la méthode de tri rapide pour créer un index, si un fichier temporaire dépasserait cette taille.

myisam_sort_buffer_size La taille du buffer lors de la restauration de table.

La restauration automatique est activée si vous lancez mysqld avec l'option **–myisam-recover**. Dans ce cas, lorsque le serveur ouvre la table MyISAM, il vérifie si la table a été marquée comme crashée ou si le compteur de tables ouvertes n'est pas zéro ou si le serveur utilise **–skip-external-locking**. Si une des conditions précédente est vraie, il arrive ceci

- La table est analysée pour rechercher des erreurs.
- Si le serveur trouve une erreur, il essaie de faire une réparation rapide (avec le tri, sans recréer de données).
- Si la réparation échoue à cause d'une erreur dans le fichier de données (par exemple, une erreur de clé), le serveur essaie à nouveau, en re-créant le fichier de données.
- Si la réparation échoue encore, le serveur essaie encore avec une ancienne méthode réparation (écrire les lignes les unes après les autres, sans tri). Cette méthode devrait être capable de réparer tout les types d'erreurs, et elle occupe peu de place sur le disque.
- Si la restauration n'est toujours pas capable de retrouver toutes les lignes, et que vous n'avez pas spécifié l'option **FORCE** dans la valeur de l'option **–myisam-recover**, la réparation automatique s'annule, avec le message d'erreur suivant : **Error : Couldn't repair table : test.g00pages**
- Si vous spécifiez la valeur **FORCE**, une alerte comme celle-ci sera écrite dans les logs : **Warning : Found 344 of 354 rows when repairing ./test/g00pages**

Notez que si la valeur de restauration automatique inclut **BACKUP**, le processus de restauration créera des fichiers avec des noms de la forme **tbl_name-datetime.BAK**. Vous devriez avoir une tâche régulière avec cron pour supprimer automatiquement ces fichiers dans les bases de données pour nettoyer le volume.

Pour avoir une approximation de la taille du fichier d'index, faire la somme de **(longueur_clef+4)/0.67** pour toutes les clefs. (c'est le pire des cas où les clefs sont insérées dans l'ordre et qu'aucune n'est compressée).

MyISAM supporte 3 différents types de tables. Deux des trois sont choisis automatiquement selon le type de colonne que vous utilisez. Le troisième, tables compressées, ne peut être crée qu'avec l'outil **myisampack**. Quand vous créez une table avec **CREATE** ou en modifiez la structure avec **ALTER** vous pouvez, pour les tables n'ayant pas de champ **BLOB** forcer le type de table en **DYNAMIC** ou **FIXED** avec l'option **ROW_FORMAT=#** des tables. Bientôt, vous pourrez compresser/décompresser les tables en spécifiant **ROW_FORMAT=compressed | default** à **ALTER TABLE**.

Tables statiques : Ceci est le format par défaut. Il est utilisé lorsque la table ne contient pas de colonnes de type **VARCHAR**, **BLOB**, ou **TEXT**. Ce format est le plus simple et le plus sûr. C'est aussi le format sur disque le plus rapide.

- Toutes les colonnes **CHAR**, **NUMERIC**, et **DECIMAL** sont complétées par des espaces jusqu'à atteindre la longueur totale de la colonne.
- Très rapide.
- Facile à mettre en cache.
- Facile à reconstruire après un crash, car les enregistrements sont localisés dans des positions fixées.
- N'a pas à être réorganisé (avec **myisamchk**) sauf si un grand nombre de lignes est effacé et que vous voulez retourner l'espace libéré au système d'exploitation.
- Requière généralement plus d'espace disque que les tables dynamiques.

Tables à format de ligne dynamiques : Ce format est utilisé avec les tables qui contiennent des colonnes de type **VARCHAR**, **BLOB** ou **TEXT**, ou si la table a été créée avec l'option **ROW_FORMAT=dynamic**. Vous pouvez utiliser la commande **SQL OPTIMIZE table** ou Shell **myisamchk** pour défragmenter une table.

- Toutes les colonnes de type chaîne sont dynamiques (hormis celle qui sont de taille inférieure à 4).

- Chaque ligne est précédée d'un octet qui indique quelles sont les lignes vides (", bit à 1) et celle qui le ne sont pas (bit à 0). Une colonne vide n'est pas la même choses qu'une colonne qui contient **NULL**. Si une colonne a une taille de zéro après avoir supprimé les espaces finaux, ou un nombre a une valeur de zéro, il est marqué dans cet octet, et la colonne sera ignorée sur le disque. Les chaînes non vides sont sauvées avec un octet de plus pour y stocker la taille.
- Ce format prend généralement moins de place que des tables à format fixe.
- Chaque ligne consomme autant d'espace que nécessaire. Si une ligne devient trop grande, elle sera coupée en blocs et écrites dans le fichier de données. Cela engendre la fragmentation du fichier de données. Dans ce cas, vous pouvez avoir à exécuter la commande **myisamchk -r** de temps en temps pour améliorer les performances. Utilisez **myisamchk -ei tbl_name** pour obtenir des statistiques.
- Ce format de table n'est pas toujours facile à reconstituer après un crash, car une ligne peut être fragmentée en de nombreux blocs, et un fragment peut manquer.

La taille d'une ligne de format variable se calcule avec

```
3
+ (nombre de colonnes + 7) / 8
+ (nombre de colonnes de tailles chars)
+ taille compactée des colonnes numériques
+ taille des chaînes
+ (nombre de colonne de valeur NULL + 7) / 8
```

- Il y a un aussi un supplément de 6 octets pour chaque lien. Une ligne de format dynamique utilise un lien à chaque fois qu'une modification cause un agrandissement de la ligne. Chaque nouveau bloc lié fait au moins 20 octets, pour que le prochain agrandissement utilise aussi ce bloc. Si ce n'est pas le cas, un nouveau bloc sera lié, avec un autre coût de 6 octets. Vous pouvez vérifier le nombre de liens dans une table avec la commande **myisamchk -ed**. Tous les liens sont supprimés avec la commande **myisamchk -r**.

tables compressées : C'est un type en lecture seule qui est généré avec l'outil optionnel **myisampack**.

- Les tables compressées prennent très peu d'espace disque.
- Chaque ligne est compressée séparément (optimisation des accès). L'en-tête d'un enregistrement est fixé (1-3 octets) selon le plus grand enregistrement dans la table. Chaque colonne est compressée différemment. Quelques un des types de compressions sont :
  - Compression des espaces en suffixe.
  - Compression des espaces en préfixe.
  - Les nombres avec la valeur 0 sont stockés en utilisant 1 octet.
  - Si les valeurs dans une colonne de type entier ont un petit intervalle, la colonne est stockée en utilisant le type le plus petit possible.
  - Si une colonne n'a qu'un petit éventail de valeurs, son type est changé en ENUM.
  - Une colonne peut utiliser une combinaison des compressions précédentes.
  - Peut gérer les enregistrements de tailles fixes ou variables.

Vous pouvez réparer une table corrompue avec **REPAIR TABLE**. Vous pouvez aussi réparer une table, lorsque **mysqld** ne fonctionne pas, avec la commande **myisamchk**

Tables assemblées MERGE

Une table **MERGE** est un groupe de tables **MyISAM** identiques qui sont utilisées comme une seule. "Identique" signifie que toutes les tables ont la même structure de colonnes et d'index

Tables MEMORY (HEAP)

Le moteur de stockage **MEMORY** crée des tables dont le contenu est stocké en mémoire.

- Vous pouvez avoir des clés non-uniques dans une table **MEMORY**. (C'est une fonctionnalité rare pour les index hash).
- Si vous avez un index hash sur une table **HEAP** avec un haut degré de duplication (de nombreux valeurs d'index contiennent la même valeur), les modifications dans cette table peuvent affecter les valeurs des clés et toutes les suppressions seront plus lentes. Le facteur de ralentissement est proportionnel au degré de duplication (ou inversement proportionnel à la cardinalité).
- Les tables **HEAP** utilisent un format de ligne fixe.
- **HEAP** ne supporte pas les colonnes de type **BLOB/TEXT**.
- **HEAP** ne supporte pas les colonnes de type **AUTO_INCREMENT**.
- Les tables **HEAP** sont partagées entre tous les clients (comme une autre table).
- La caractéristique des tables **MEMORY** qui fait que les tables sont stockées en mémoire est partagée avec les tables internes que le serveur crée à la volée lors du traitement des requêtes. Cependant, les tables internes ont aussi la capacité d'être converties en tables disques automatiquement, si elles deviennent trop grandes. La taille limite est déterminée par la valeur de **tmp_table_size**.

Les tables **MEMORY** ne peuvent pas être converties en tables disques. Pour vous assurer que vous ne faîtes rien de dangereux pour le serveur, vous pouvez utiliser la variable système **max_heap_table_size** pour imposer une taille maximale aux tables **MEMORY**. Pour des tables individuelles, vous pouvez utiliser l'option de table **MAX_ROWS** avec la commande **CREATE TABLE**.

- Vous avez besoin de suffisamment de mémoire pour accepter toutes les tables **HEAP** que vous allez utiliser simultanément.
- Pour libérer de la mémoire, vous devez exécuter la commande **DELETE FROM heap_table**, **TRUNCATE heap_table** ou **DROP TABLE heap_table**.
- Si vous voulez remplir les tables **MEMORY** au lancement du serveur MySQL, vous pouvez utiliser l'option **–init-file**. Par exemple, vous pouvez mettre les commandes telles que **INSERT INTO ... SELECT** et **LOAD DATA INFILE** pour lire des données dans une source de données persistante.
- Si vous utilisez la réplication, les tables **MEMORY** du maître se vident à l'extinction. Cependant, un esclave peut ne pas s'apercevoir que ces tables ont été vidées, et il risque de retourner des données invalides si vous l'utilisez. lorsqu'une table **MEMORY** est utilisée sur le maître, il émet une commande **DELETE FROM** automatiquement, pour synchroniser l'esclave et le maître. Notez que même avec cette stratégie, l'esclave aura des données obsolètes entre le moment où le maître s'éteint et celui où il est redémarré. Mais si vous utilisez l'option **–init-file** pour remplir la table **MEMORY** au lancement du serveur, elle s'assurera que cette intervalle est bien null.

La mémoire nécessaire pour les tables HEAP sont
SUM_OVER_ALL_KEYS(max_length_of_key + sizeof(char*) * 2)
+ ALIGN(length_of_row+1, sizeof(char*))

ALIGN() représente un facteur d'arrondi, car la taille de la ligne doit faire exactement un multiple de la taille du pointeur de char. **sizeof(char*)** vaut 4 sur les machines 32 bits et 8 sur une machine 64 bits.

Tables BDB ou BerkeleyDB

En utilisant les tables **BDB**, vos tables ont plus de chances de survivre aux crashs, et vous avez accès à **COMMIT** et **ROLLBACK** avec les transactions. Les options suivantes de mysqld peuvent être utilisées pour modifier le comportement des tables BDB :

–bdb-home=répertoire Répertoire de base des tables BDB. Cela doit être le même répertoire que vous avez utilisés pour –datadir.

–bdb-lock-detect=# Détection des verrouillages Berkeley. (DEFAULT, OLDEST, RANDOM, ou YOUNGEST).

–bdb-logdir=répertoire Répertoire des fichiers de log de Berkeley DB.

–bdb-no-recover Ne pas démarrer Berkeley DB en mode de restauration.

–bdb-no-sync Ne pas vider les tampons synchroniquement.

–bdb-shared-data Démarrer Berkeley DB en mode multi-processus (Ne pas utiliser DB_PRIVATE lors de l'initialisation de Berkeley DB)

–bdb-tmpdir=répertoire Répertoire des fichiers temporaires de Berkeley DB.

–skip-bdb Désactive l'utilisation des tables BDB.

Les variables systèmes suivante affectent le comportement des tables BDB. **bdb_max_lock** est le nombre maximal de verrous actifs sur une table BDB. Si vous utilisez **–skip-bdb**, MySQL n'initialisera pas la bibliothèque Berkeley DB et cela économisera beaucoup de mémoire. Bien sûr, vous ne pouvez pas utiliser les table BDB si vous utilisez cette option. Si vous essayez de créer une table BDB, MySQL créera une table MyISAM à la place.

Normalement, vous devez démarrer mysqld sans **–bdb-no-recover** si vous avez l'intention d'utiliser des tables BDB. Cela peut cependant vous poser des problèmes si vous essayez de démarrer mysqld alors que des fichiers de log BDB sont corrompus. Vous pouvez spécifier le nombre maximal de verrous avec **bdb_max_lock** (10000 par défaut) que vous pouvez activer sur une table BDB. Vous devez l'augmenter si vous obtenez des erreurs du type : **bdb : Lock table is out of available locks Got error 12 from ...** lorsque vous avez fait de longues transactions ou quand mysqld doit examiner beaucoup de lignes pour calculer la requête. Vous pouvez aussi changer les options **binlog_cache_size** et **max_binlog_cache_size** si vous utilisez de grandes transactions multi-lignes.

Moteur de table EXAMPLE

C'est un moteur "bidon" qui ne fait rien du tout. Son but est de fournir des exemples au niveau du code source de MySQL pour illustrer l'écriture d'un moteur de table. En tant que tel, il intéressera surtout les développeurs.

Moteur de table FEDERATED

C'est un moteur de table qui accède à des tables dans une base de données distante, plutôt que dans des fichiers locaux.

Moteur de table ARCHIVE

Il est utilisé pour stocker de grande quantité de données, sans index, et de manière très économique. Le moteur **ARCHIVE** ne supporte que les commandes **INSERT** et **SELECT** : aucun effacement, remplacement ou modification. Une commande **SELECT** effectue un scan de table complet. Les enregistrements sont compressés au moment de leur insertion. Vous pouvez utiliser la commande **OPTIMIZE TABLE** pour analyser la table, et compresser encore plus.

Moteur CSV

Ce moteur stocke les données dans un fichier texte, avec le format valeurs séparées par des virgules.

Moteur de table InnoDB

InnoDB fournit à MySQL un gestionnaire de table transactionnelle (compatible ACID), avec validation (commits), annulations (rollback) et capacités de restauration après crash. **InnoDB** utilise un verrouillage de lignes, et fournit des lectures cohérentes comme Oracle, sans verrous. Ces fonctionnalités accroissent les possibilités d'utilisation simultanées des tables, et les performances. Il n'y a pas de problème de queue de verrous avec InnoDB, car les verrous de lignes utilisent très peu de place. Les tables InnoDB sont les premières tables MySQL qui supportent les contraintes de clés étrangères (**FOREIGN KEY**).

Si vous ne souhaitez pas utiliser les tables **InnoDB**, vous pouvez ajouter l'option **skip-innodb** dans votre fichier d'options MySQL. Les deux ressources disques importantes gérées par InnoDB sont sa table de données et son fichier de log. Si vous ne spécifiez aucune options de configuration InnoDB, MySQL créera un fichier de données auto-croissant appelé **ibdata1** et deux fichiers de log de 5 Mo appelés

ib_logfile0 et **ib_logfile1** dans le dossier de données MySQL.

Pour configurer le fichier de données InnoDB, utilisez l'option innodb_data_file_path dans my.cnf :
innodb_data_file_path=datafile_spec1[;datafile_spec2]...

Exemples

Cette configuration crée un fichier de données de 10 Mo ibdata1, auto-croissant. Il n'y a pas de dossier de sauvegarde d'indiqué : par défaut, c'est le dossier de données de MySQL.
[mysqld]
innodb_data_file_path=ibdata1 :10M :autoextend
Une table contenant 50 Mo de données, appelée ibdata1 et un fichier 50 Mo auto-croissant, appelé ibdata2 dans le dossier de données
[mysqld]
innodb_data_file_path=ibdata1 :50M ;ibdata2 :50M :autoextend
La syntaxe complète de la spécification de fichier de données inclut le nom du fichier, sa taille, et différents attributs :
file_name :file_size[:autoextend[:max :max_file_size]]
[mysqld]
innodb_data_file_path=ibdata1 :10M :autoextend :max :500M
La ligne de configuration suivante permet au fichier ibdata1 de croître jusqu'à 500 Mo
[mysqld]
innodb_data_home_dir = /ibdata
innodb_data_file_path=ibdata1 :50M ;ibdata2 :50M :autoextend
crée deux fichiers appelés ibdata1 et ibdata2 mais les place dans le dossier /ibdata,
!!!!Note : InnoDB ne crée pas les dossiers
Si vous spécifier l'option innodb_data_home_dir sous forme de chaîne vide, vous pouvez spécifier des noms de chemins absolus dans la valeur de innodb_data_file_path. L'exemple ci-dessous est équivalent au précédent :
[mysqld]
innodb_data_home_dir =
innodb_data_file_path=/ibdata/ibdata1 :50M ;/ibdata/ibdata2 :50M :autoextend
Exemple de configuration

```
[mysqld]
# Vous pouvez placer d'autres options MYSQL ici
# ...
# Le fichier de données doit contenir vos données et index.
# Assurez vous que vous avez l'espace disque nécessaire.
innodb_data_file_path = ibdata1:10M:autoextend
#
# Utilisez un buffer de taille 50 à 80 % de votre mémoire serveur
set-variable = innodb_buffer_pool_size=70M
set-variable = innodb_additional_mem_pool_size=10M
#
# Utilisez un fichier de log de taille 25 % du buffer mémoire
set-variable = innodb_log_file_size=20M
set-variable = innodb_log_buffer_size=8M
#
innodb_flush_log_at_trx_commit=1
```

Options de démarrage InnoDB

innodb_additional_mem_pool_size La taille du buffer mémoire d'InnoDB, pour ses dictionnaires d'informations, et ses structures internes de données. Une valeur pratique est 2Mo, mais plus vous aurez de tables dans votre application, plus vous devrez augmenter cette valeur. Si InnoDB est à court de mémoire, il va allouer de la mémoire auprès du système, et écrire des messages dans le fichier de logs.

innodb_buffer_pool_size La taille de buffer mémoire que InnoDB utiliser pour mettre en cache les données et les index de tables. Plus cette valeur est grand, et moins vous ferez d'accès disques. Sur un serveur dédiés, vous pouvez monter cette valeur jusqu'à 80% de la mémoire physique de la machine. Ne lui donnez pas une valeur trop grande, car cela peut engendrer l'utilisation de mémoire sur le disque par votre serveur.

innodb_data_file_path Chemin individuel vers les fichiers de données, et leur taille. Le chemin complet de chaque fichier de données est créé en concaténant **innodb_data_home_dir** avec les chemins spécifiés ici. La taille du fichier est spécifiée en méga-octets, ce qui explique la présence du '**M**' après les spécifications ci-dessus. Vous pouvez donner au fichier une taille supérieure à 4 Go sur les systèmes d'exploitation qui acceptent les gros fichiers.

Sur certains systèmes, la taille doit être inférieure à 2 Go.Si vous ne spécifiez pas **innodb_data_file_path**, le comportement par défaut est de créer un fichier auto-croissant de 10 Mo, appelé **ibdata1**. vous pouvez donner une taille de fichier de plus de 4Go sur les systèmes d'exploitation qui supportent les grands fichiers. Vous pouvez aussi utiliser les partition raw.

innodb_data_home_dir La partie commune du chemin de tous les fichiers de données InnoDB. Si vous ne mentionnez pas cette option, la valeur par défaut sera celle du dossier de données MySQL. Vous pouvez aussi spécifier une chaîne vide, et dans ce cas, les chemins spécifiés dans **innodb_data_file_path** seront des chemins absolus.

innodb_fast_shutdown Par défaut, InnoDB fait une purge complète et vide le buffer d'insertion avant une extinction. Ces opérations peuvent prendre beaucoup de temps. Si vous donnez à ce paramètre la valeur de 1, InnoDB ignore ces opérations d'extinction. Sa valeur par défaut est 1.

innodb_file_io_threads Nombre de pointeurs de fichier de InnoDB. Normalement, cette valeur doit être de 4.

innodb_file_per_table Cette option fait que InnoDB va stocker chaque table dans un fichier .ibd indépendant. Voyez la section sur les espaces de tables multiples.

innodb_flush_log_at_trx_commit Normalement, cette option vaut 1, ce qui signifie que lors de la validation de la transaction, les logs sont écrits sur le disque, et les modifications faites par la transaction deviennent permanentes, et survivront un crash de base. Si vous souhaitez réduire la sécurité de vos données, et que vous exécutez de petites transactions, vous pouvez donner une valeur de 0 à cette option, pour réduire les accès disques.

innodb_flush_method La valeur par défaut pour cette option est fdatasync. Une autre option est O_DSYNC.

innodb_force_recovery Attention : cette option ne doit être définie que dans les cas où vous voulez exporter les données d'une base corrompue, dans une situation d'urgence. Les valeurs possibles de cette option vont de 1 à 6. Par mesure de sécurité, InnoDB empêche les modifications de données si la valeur de cette option est supérieure à 0.

innodb_lock_wait_timeout Le délai d'expiration des transactions InnoDB, en cas de blocage de verrou, avant d'annuler. InnoDB détecte automatiquement les blocages de verrous et annule alors les transactions. Si vous utilisez la commande LOCK TABLES, ou un autre gestionnaire de table transactionnelles que InnoDB dans la même transaction, un blocage de verrou peut survenir, et InnoDB ne pourra pas le détecter. Ce délai est donc pratique pour résoudre ces situations.

innodb_log_arch_dir Le dossier où les logs complétés doivent être archivés, si nous utilisons l'archivage de logs. La valeur de ce paramètre doit être actuellement la même que la valeur de **innodb_log_group_home_dir**.

innodb_log_archive Cette valeur doit être actuellement de 0. Au moment de la restauration de données à partir d'une sauvegarde, à l'aide des log binaires de MySQL, il n'y a actuellement pas besoin d'archiver les fichiers de log InnoDB.

innodb_log_buffer_size La taille du buffer que InnoDB utilise pour écrire les log dans les fichiers de logs, sur le disque. Les valeurs utiles vont de 1 Mo à 8 Mo. Un grand buffer de log permet aux grandes transactions de s'exécuter sans avoir à écrire de données dans le fichier de log jusqu'à la validation. Par conséquent, si vous avez de grandes transactions, augmenter cette taille va réduire les accès disques.

innodb_log_file_size Taille de chaque fichier de log dans un groupe de log, exprimé en méga-octets. Les valeurs pratiques vont de 1Mo à une fraction de la taille du buffer de log (1 / le nombre de logs, en fait). Plus la taille est grande, moins de points de contrôles seront utilisés, réduisant les accès disques. La taille combinée des logs doit être inférieure à 4 Go sur les systèmes 32 bits.

innodb_log_files_in_group Nombre de fichiers de logs dans le groupe de log. InnoDB écrit dans ces fichiers de manière circulaire. Une valeur de 2 est recommandée. C'est la valeur par défaut.

innodb_log_group_home_dir Le dossier pour les fichiers de logs. Il doit avoir la même valeur que **innodb_log_arch_dir**. Si vous ne spécifiez pas de paramètre de log InnoDB, la configuration par défaut va créer deux fichiers de logs de 5 Mo, appelés ib_logfile0 et ib_logfile1 dans le dossier de données MySQL.

innodb_max_dirty_pages_pct Cette entier va de 0 à 100. Par défaut, il vaut 90. Le thread principal de InnoDB essaie de transmettre les pages au pool de buffer, pour qu'un pourcentage maximal de **innodb_max_dirty_pages_pct** soit encore en attente de flush. Si vous avez le droit de SUPER, ce pourcentage peut être changée durant l'exécution du serveur :**SET GLOBAL innodb_max_dirty_pages_pct = value ;**

innodb_mirrored_log_groups Nombre de copies identiques de groupe de log que nous conservons. Actuellement, cette valeur doit être au minimum de 1.

innodb_open_files Ce n'est utile que si vous utilisez les espaces de tables multiples. Cette option spécifie que le nombre maximal de fichier .ibd que InnoDB peut garder ouvert simultanément. La valeur minimum est de 10. La valeur maximum est de 300.

Les pointeurs de fichiers utilisés par .ibd sont réservés pour InnoDB. Ils sont indépendants de ceux spécifiés par —open-files-limit, et
innodb_thread_concurrency

InnoDB essaie de garder le nombre de thread système concurents inférieur à la limite de ce paramètre. La valeur par défaut est 8. Si

vous pouvez essayer d'augmenter la valeur, pour utiliser au mieux les ressources disponibles. Une valeur recommandée est la somme

Créer des bases InnoDB

utiliser TYPE = InnoDB. ex :
CREATE TABLE CUSTOMER (A INT, B CHAR (20), INDEX (A)) TYPE = InnoDB ;
La quantité d'espace disponible apparaît dans la section de commentaire de la commande SHOW. Par exemple :
SHOW TABLE STATUS FROM test LIKE 'CUSTOMER'
vous pouvez désactiver l'auto-validation avec la commande SET AUTOCOMMIT = 0 et utiliser les commandes COMMIT et ROLLBACK pour valider ou annuler vos transactions. Si vous voulez laisser le mode d'auto-validation tranquille, vous pouvez placer vos commandes entre START TRANSACTION et COMMIT ou ROLLBACK.

Convertir des tables MyIsam vers InnoDB

pour utiliser le moteur InnoDB pour la création de table par défaut, utiliser **default-table-type=innodb** dans le groupe **[mysqld]**. Le moyen le plus rapide pour mettre la table au format InnoDB est d'insérer directement les lignes dans une table InnoDB, c'est à dire, utiliser **ALTER TABLE ... TYPE=INNODB**, ou créer un table InnoDB vide, avec la même définition et de faire l'insertion de toutes les lignes avec **INSERT INTO ... SELECT * FROM ....**

INSERT INTO newtable SELECT * FROM oldtable
WHERE yourkey > something AND yourkey <= somethingelse ;
Une fois que toutes les données ont été insérées dans la table, vous pouvez la renommer.
Colonnes AUTO_INCREMENT avec InnoDB : Innodb va ajouter dans le dictionnaire de données un compteur spécial appelé compteur auto-increment, qui est utilisé pour assigner les nouvelles valeurs de la colonne.

Contraintes de clés étrangères FOREIGN KEY : La syntaxe des définitions de contraintes de clés étrangères de InnoDB est la suivante :

[CONSTRAINT symbol] FOREIGN KEY [id] (index_col_name, ...)
REFERENCES tbl_name (index_col_name, ...)
[ON DELETE CASCADE | SET NULL | NO ACTION | RESTRICT]
[ON UPDATE CASCADE | SET NULL | NO ACTION | RESTRICT]
Les deux tables doivent être de type InnoDB, dans la table, il doit y avoir un **INDEX** où les clés étrangères sont listées comme première colonne, dans le même ordre, et dans la table référencée, il doit y avoir un **INDEX** où les colonnes référencées sont listées comme premières colonnes, dans le même ordre. Les préfixes d'index ne sont pas supportés pour les clés de contrainte.

exemple :
```
CREATE TABLE parent(id INT NOT NULL,
PRIMARY KEY (id)
) TYPE=INNODB ;
CREATE TABLE child(id INT, parent_id INT,
INDEX par_ind (parent_id),
FOREIGN KEY (parent_id) REFERENCES parent(id)
ON DELETE CASCADE
) TYPE=INNODB ;
exemple plus complexe :
CREATE TABLE product (category INT NOT NULL, id INT NOT NULL,
```

```
price DECIMAL,
PRIMARY KEY(category, id)) TYPE=INNODB ;
CREATE TABLE customer (id INT NOT NULL,
PRIMARY KEY (id)) TYPE=INNODB ;
CREATE TABLE product_order (no INT NOT NULL AUTO_INCREMENT,
product_category INT NOT NULL,
product_id INT NOT NULL,
customer_id INT NOT NULL,
PRIMARY KEY(no),
INDEX (product_category, product_id),
FOREIGN KEY (product_category, product_id)
REFERENCES product(category, id)
ON UPDATE CASCADE ON DELETE RESTRICT,
INDEX (customer_id),
FOREIGN KEY (customer_id)
REFERENCES customer(id)) TYPE=INNODB ;
```

InnoDB et la réplication

Il est aussi possible d'utiliser la réplication pour que les tables de l'esclave ne soient pas les mêmes que les tables du maître. Par exemple, vous pouvez répliquer les modifications d'une table **InnoDB** sur le maître dans une table **MyISAM** sur l'esclave. Pour configurer un nouvel esclave sur le maître, vous devez faire une copie de l'espace de table InnoDB, des fichiers de log, ainsi que les fichiers .frm des tables InnoDB, et les placer sur l'esclave. Il y a des limitations mineures à la réplication **InnoDB** : **LOAD TABLE FROM MASTER** ne fonctionne pas pour les tables InnoDB. Il y a des palliatifs : 1) exportez la table du maître, et envoyez la sur l'esclave, ou, 2) utilisez **ALTER TABLE tbl_name TYPE=MyISAM** sur le maître avant de configurer la réplication avec **LOAD TABLE tbl_name FROM MASTER**, et ensuite, **ALTER TABLE** pour remettre les tables en mode InnoDB après cela.