Administration MySQL

Exportation des données en CSV

Pour exporter les données en CSV, il est possible de préciser à la sortie d'une requete SQL, le OUTPUT FILE
Mais il faut préciser dans le fichier my.cnf un répertoire sécurisé:

```
mysql> select * from myidtables INTO OUTFILE 'myids.csv' ;
ERROR 1290 (HY000): The MySQL server is running with the --secure-file-priv option so it
cannot execute this statement
```

Dans le fichier my.cnf, rajouter le repertoire de sauvegarde des données:

```
secure_file_priv=/tmp
```

Une fois signifier, les données peuvent se faire sauvegardé dans le repertoire donné.

```
mysql> SHOW VARIABLES LIKE "secure_file_priv";
+------------------+-------+
| Variable_name | Value |
+------------------+-------+
| secure_file_priv | /tmp/ |
+------------------+-------+
1 row in set (0,00 sec)

mysql> select * from myidtables INTO OUTFILE '/tmp/myids.csv' ;
Query OK, 1 row affected (0,00 sec)
```

Il est possible de signifier le caractère de terminaison inter valeur ainsi que le caractère entourant les valeurs

```
select * from myidtables INTO OUTFILE '/tmp/myids2.csv' FIELDS ENCLOSED BY '"' TERMINATED BY
';' ;
```

Importation des données

Il est possible de reimporter des données CSV dans une table mysql.
En premier lieu, nous créons une nouvelle table

```
mysql> create table import( i int);
Query OK, 0 rows affected (0,06 sec)
```

Puis nous utilisons la requete LOAD DATA INFILE '''nom de fichier''' into table '''table name'''

```
mysql> LOAD DATA INFILE '/tmp/myids2.csv' into table import FIELDS ENCLOSED BY '"' TERMINATED
BY ';' ;
Query OK, 1 row affected (0,12 sec)
Records: 1 Deleted: 0 Skipped: 0 Warnings: 0
```

Exportation de données avec des en-têtes de colonne

Il serait pratique que le fichier CSV contienne la première ligne comme en-tête de colonne afin que le fichier soit plus compréhensible.

Pour ajouter les en-têtes de colonne, vous devez utiliser l'instruction UNION comme suit:

```
mysql> select 'key' union select * from myidtables INTO OUTFILE '/tmp/myids3.csv' FIELDS
ENCLOSED BY '"' TERMINATED BY ';' ;
```

Donne comme fichier

```
<source lang='bash'>
"key"
"1"
</source>
```

Tuning MySQL

innodb_file_per_table

Cette option permet d'avoir 1 fichier par table innoDB au lieu d'un seul fichier

innodb_file_per_table=ON

innodb_stats_on_metadata

Lorsque l'option est définie sur ON, les statistiques d'index InnoDB sont mises à jour lors de l'exécution de SHOW TABLE STATUS, de SHOW INDEX ou de la requête d'INFORMATION_SCHEMA.TABLES ou d'INFORMATION_SCHEMA.STATISTICS. Ces statistiques incluent la cardinalité et le nombre d'entrées. Elles sont utilisées par l'optimiseur pour trouver un plan d'exécution optimal.

Ainsi, même si les instructions SELECT ne peuvent pas modifier les statistiques réelles, MySQL met à jour les statistiques des tables InnoDB. C'est contre-intuitif.

Est-ce utile? Pas vraiment, car InnoDB calculera toujours des statistiques lorsque vous ouvrez une table pour la première fois et lorsque des parties importantes de la table ont été modifiées (et lorsque vous exécutez ANALYZE TABLE).

Maintenant, pourquoi avons-nous eu une charge de lecture aussi élevée lorsque innodb_stats_on_metadata a été défini sur ON? Pour InnoDB, les statistiques sont estimées à partir de plongées à index aléatoire, ce qui se traduit par des lectures aléatoires.

innodb_buffer_pool_instances

Diviser le pool mémoire en plusieurs "régions" peut permettre d’améliorer l’accès aux données de façon concurrente, de diminuer les conflits issus des threads lors d’opérations de lecture ou d’écriture sur des pages verrouillées.

Chaque pool utilise alors sa propre liste d’accès aux données, sa liste de "flush", son propre mécanisme de LRU et surtout son propre buffer "Mutex".

Tous ces éléments rendent chaque région indépendante et permettent de diminuer les phénomènes de contention.

Ce paramètre n’est efficace que si la définition mémoire du innodb_buffer_pool_size est supérieure à 1 Go.

innodb_change_buffer_max_size

La variable innodb_change_buffer_max_size permet de configurer la taille maximale de la mémoire tampon de changement sous forme de pourcentage de la taille totale du pool de mémoire tampon. Par défaut, innodb_change_buffer_max_size est défini sur 25. Le paramètre maximal est 50.

Pensez à augmenter innodb_change_buffer_max_size sur un serveur MySQL avec une activité importante d'insertion, de mise à jour et de suppression, où la fusion de tampons de modification ne suit pas le rythme des nouvelles entrées de tampon de modifications, ce qui lui permet d'atteindre sa taille maximale.

Envisagez de réduire innodb_change_buffer_max_size sur un serveur MySQL avec des données statiques utilisées pour la génération de rapports, ou si la mémoire tampon de modification utilise trop d’espace mémoire partagé avec le pool de mémoire tampon, ce qui entraîne la disparition plus rapide des pages du pool de mémoire tampon.

query_cache_type

Sert de cache au requete coté serveur. A desactiver, les performances sont mauvaise.

Sinon mettre le query_cache_type à 2

query_cache_type = 1
query_cache_size = 256M

query_cache_type est le type de cache que l’on va adopter:
0 = pas de cache
1 = met en cache toutes les requetes sauf celles qui ont le flag “SELECT S_NO_CACHE”
2 = met en cache seulement les requetes qui comportent le flag “SELECT SQL_CACHE”

innodb_flush_method

La bonne valeur pour éviter un double buffering est O_DIRECT

SQL Mode

Le serveur MySQL peut fonctionner dans différents modes SQL et peut appliquer ces modes différemment pour différents clients, en fonction de la valeur de la variable système sql_mode. Les administrateurs de base de données peuvent définir le mode SQL global pour répondre aux exigences opérationnelles du serveur de site. Chaque application peut définir son mode de session SQL en fonction de ses propres exigences.

- ALLOW_INVALID_DATES

N'effectuez pas de vérification complète des dates. Vérifiez uniquement que le mois est compris entre 1 et 12 et le jour entre 1 et 31.

- ANSI_QUOTES

Traitez "comme un caractère de citation d’identificateur (comme le` caractère de citation) et non comme un caractère de citation de chaîne.

- ERROR_FOR_DIVISION_BY_ZERO

Le mode ERROR_FOR_DIVISION_BY_ZERO affecte le traitement de la division par zéro, ce qui inclut MOD (N, 0). Pour les opérations de changement de données (INSERT, UPDATE), son effet dépend également de l'activation du mode SQL strict.

- ◦ Si ce mode n'est pas activé, la division par zéro insère NULL et ne produit aucun avertissement.
- ◦ Si ce mode est activé, la division par zéro insère NULL et génère un avertissement.

- HIGH_NOT_PRECEDENCE

Définiie la précedence de l'opérateur NOT .

- NO_BACKSLASH_ESCAPES

Désactive l'utilisation du caractère barre oblique inverse (\) en tant que caractère d'échappement dans les chaînes. Lorsque ce mode est activé, la barre oblique inverse devient un caractère ordinaire comme un autre.

- NO_ENGINE_SUBSTITUTION

Contrôle la substitution automatique du moteur de stockage par défaut lorsqu'une instruction telle que CREATE TABLE ou ALTER TABLE spécifie un moteur de stockage désactivé ou non compilé.

- ANSI

Équivalent à REAL_AS_FLOAT, PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE et ONLY_FULL_GROUP_BY.

- TRADITIONAL

Equivalent à STRICT_TRANS_TABLES, STRICT_ALL_TABLES, NO_ZERO_IN_DATE, NO_ZERO_DATE, ERROR_FOR_DIVISION_BY_ZERO et NO_ENGINE_SUBSTITUTION.

Configuration conseillé

Dans le fichier my.ini initialisé sql_mode a ANSI et TRADITIONAL. Ces parametres renforce la sécurité et une proximité avec Oracle <source lang='ini'> sql_mode = "ANSI,TRADITIONAL" </source>

Une fois redemarré, le nouveaux SQL mode des clients est homogène à <source lang='SQL'>

</source>

```
+--------------
+-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
----------------------+
|  Variable_name |  Value |+--------------
+-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----+--------------+
| sql_mode |
REAL_AS_FLOAT, PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, ONLY_FULL_GROUP_BY, ANSI, STRICT_TRANS_TA
BLES, STRICT_ALL_TABLES, NO_ZERO_IN_DATE, NO_ZERO_DATE, ERROR_FOR_DIVISION_BY_ZERO, TRADITIONAL, NO_
ENGINE_SUBSTITUTION |
+--------------
+-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
---------------------+ 1 row in set (0,04 sec)
```

Test SQL Mode

la variable sql_mode permet de savoir le mode sql par defaut du serveur

```
mysql> SHOW SESSION VARIABLES LIKE 'sql_mode';
+--------------
+---------------------------------------------------------------------------------------------
--------------------------+
| Variable_name |  Value |
+--------------
+---------------------------------------------------------------------------------------------
--------------------------+
| sql_mode |
ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO
,NO_ENGINE_SUBSTITUTION |
+--------------
+---------------------------------------------------------------------------------------------
--------------------------+
1 row in set (0,00 sec)
```

Il est possible de tester le mode et d'override pour la session la prise en compte du SQL

```
mysql> create table test(i INT); Query OK, 0 rows affected (0,12 sec)

mysql> insert into test values (2/0); ERROR 1365 (22012): Division by 0

mysql> set sql_mode=; Query OK, 0 rows affected (0,00 sec)

mysql> insert into test values (2/0); Query OK, 1 row affected (0,06 sec)
```

Migration

Migration

La facon la plus simple est de faire un dump des données en SQL via mysqldump et de reinjecter les données dans une base vierge. Malheuresement ce n'est pas forcement possible du au fait de la taille des données. Nous allons essayer de faire une migration directe

InnoDB Fast Shutdown

Le mode d'arrêt InnoDB. Si la valeur est 0, InnoDB effectue un arrêt lent, une purge complète et une fusion de la mémoire tampon de modification avant la fermeture. Si la valeur est 1 (valeur par défaut), InnoDB ignore ces opérations à l’arrêt, processus appelé arrêt rapide. Si la valeur est 2, InnoDB vide ses journaux et s’arrête, comme si MySQL s’était écrasé; aucune transaction validée n'est perdue, mais l'opération de récupération sur incident prend plus de temps au prochain démarrage.

L'arrêt lent peut prendre des minutes, voire des heures dans les cas extrêmes, où d'importantes quantités de données sont encore mises en mémoire tampon. Utilisez la technique d'arrêt lent avant de mettre à niveau ou de rétrograder les versions majeures de MySQL, afin que tous les fichiers de données soient entièrement préparés au cas où le processus de mise à niveau met à jour le format de fichier. Il est possible de voir la variable via

```
SHOW SESSION VARIABLES LIKE 'innodb_fast%';
+----------------------+-------+
| Variable_name |  Value |
+----------------------+-------+
| innodb_fast_shutdown | 1 |
+----------------------+-------+
1 row in set (0,02 sec)
```

et positionner la valeur à 0

```
mysql> set global innodb_fast_shutdown=0; Query OK, 0 rows affected (0,00 sec)
```

Migration

1. Passer le maximum de table InnoDB qui ont des capacités auto reparatrice
2. Bloquer les connections entrantes via iptables
3. executer mysqlcheck

```
pilou@lubuntu:~/mysql80/mysql-8.0.13-linux-glibc2.12-x86_64$ ./bin/mysqlcheck --defaults-file=/home/pilou/mysql80/mysql-8.0.13-linux-glibc2.12-x86_64/my.cnf -u root -h localhost -p --all-databases --check-upgrade
Enter password:
crm.customer OK
mysql.columns_priv OK
mysql.component OK
mysql.db OK
mysql.default_roles OK
mysql.engine_cost OK
mysql.func OK
mysql.general_log OK
mysql.global_grants OK
mysql.gtid_executed OK
mysql.help_category OK
mysql.help_keyword OK
mysql.help_relation OK
mysql.help_topic OK
mysql.innodb_index_stats OK
mysql.innodb_table_stats OK
mysql.password_history OK
mysql.plugin OK
mysql.procs_priv OK
mysql.proxies_priv OK
mysql.role_edges OK
mysql.server_cost OK
mysql.servers OK
mysql.slave_master_info OK
mysql.slave_relay_log_info OK
mysql.slave_worker_info OK
mysql.slow_log OK
```

```
mysql.tables_priv OK
mysql.time_zone OK
mysql.time_zone_leap_second OK
mysql.time_zone_name OK
mysql.time_zone_transition OK
mysql.time_zone_transition_type OK
mysql.user OK
sqlmode.test OK
sys.sys_config OK
testdb.import OK
testdb.myidtables OK
testdb.t2 OK
testdb.t4 OK
```

Transaction

Une transaction, c'est un ensemble de requêtes qui sont exécutées en un seul bloc. Ainsi, si une des requêtes du bloc échoue, on peut décider d'annuler tout le bloc de requêtes (ou de quand même valider les requêtes qui ont réussi). On peut donc considérer que chaque requête constitue une transaction qui est automatiquement commitée. Par défaut, MySQL est donc en mode "autocommit".

Pour quitter ce mode, il suffit de lancer la requête suivante :

```
set autocommit=0;
Query OK, 0 rows affected (0,00 sec)
```

Inserons une donnée dans une table

```
mysql> use transaction;
Database changed
mysql> create table testtransaction(i integer) engine='INNODB';
Query OK, 0 rows affected (0,09 sec)
```

Connection 1	Connection 2

`mysql> begin;`
`Query OK, 0 rows affected (0,00 sec)`
`mysql> insert into testtransaction values (2);`
`Query OK, 1 row affected (0,00 sec)`
`mysql> select * from testtransaction;`
	`mysql> select * from testtransaction;`
	
	`Empty set (0,01 sec)`

Lors du commit:

Connection 1	Connection 2

`mysql> commit ;`
`Query OK, 0 rows affected (0,10 sec)`
	`select * from testtransaction;`
	`+------+`
	`| i    |`
	`+------+`
	`|    2 |`
	`+------+`
	`1 row in set (0,00 sec)`

L'operation de rollback permet d'annuler une transaction:

```
mysql> begin;
Query OK, 0 rows affected (0,00 sec)

mysql> insert into testtransaction values (5);
Query OK, 1 row affected (0,00 sec)

mysql> select * from testtransaction;
+------+
| i    |
+------+
|    2 |
|    5 |
+------+
2 rows in set (0,00 sec)

mysql> rollback;
Query OK, 0 rows affected (0,01 sec)

mysql> select * from testtransaction;
+------+
| i    |
+------+
|    2 |
+------+
1 row in set (0,00 sec)
```

Il faut faire attention que les comit/rollback en concerne pas les opération ddl (create table, alter ...)

Connection 1	Connection 2
mysql> begin -> ; Query OK, 0 rows affected (0,00 sec) mysql> alter table testtransaction add column testcolumn integer; Query OK, 0 rows affected (0,12 sec) Records: 0 Duplicates: 0 Warnings: 0	mysql> select * from testtransaction; +------+------------+ | i | testcolumn | +------+------------+ | 2 | NULL | +------+------------+ 1 row in set (0,00 sec)

Lors du rollback:

Connection 1	Connection 2
mysql> rollback ; Query OK, 0 rows affected (0,00 sec)	mysql> select * from testtransaction; +------+------------+ | i | testcolumn | +------+------------+ | 2 | NULL | +------+------------+ 1 row in set (0,00 sec)

Transaction Level

Il est possible de savoir le niveau des transaction via:

```
-- mysql> show variables like 'tx_isolation';
-- sous mysql 8 tx_isolation a été renommé en transation_isolation
mysql> show  variables like 'transaction_isolation';
+-----------------------+-----------------+
| Variable_name         | Value           |
+-----------------------+-----------------+
| transaction_isolation | REPEATABLE-READ |
+-----------------------+-----------------+
1 row in set (0,00 sec)
```

Repeatable Read

C'est le niveau d'isolation par défaut pour InnoDB. Des lectures cohérentes dans la même transaction lisent l'instantané établi par la première lecture. Cela signifie que si vous émettez plusieurs instructions SELECT simples (non verrouillables) dans la même transaction, ces instructions SELECT sont également cohérentes.

Connection 1	Connection 2
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ; Query OK, 0 rows affected (0,00 sec) mysql> create table transactionlevel (i integer); Query OK, 0 rows affected (0,12 sec) mysql> begin -> ; Query OK, 0 rows affected (0,00 sec) mysql> insert into transactionlevel values (2); Query OK, 1 row affected (0,00 sec)	mysql> SET TRANSACTION ISOLATION LEVEL REPEATABLE READ; Query OK, 0 rows affected (0,00 sec) mysql> begin; Query OK, 0 rows affected (0,00 sec) mysql> select * from transactionlevel; Empty set (0,00 sec)

Apres le commit de la première connexion, on obtient:

Connection 1	Connection 2
commit; Query OK, 0 rows affected (0,06 sec) mysql> select * from transactionlevel; +------+ | i | +------+ | 2 | +------+ 1 row in set (0,00 sec)	mysql> select * from transactionlevel; Empty set (0,00 sec)

Il faut attendre le commit de la transaction pour voir les changements;

```
mysql> select * from transactionlevel;
Empty set (0,00 sec)

mysql> commit;
Query OK, 0 rows affected (0,00 sec)

mysql> select * from transactionlevel;
+------+
| i    |
+------+
|    2 |
+------+
1 row in set (0,00 sec)
```

Table Lock vs Row Lock

Avantages du verrouillage au niveau de la ligne:

- Moins de conflits de verrous lors de l'accès à différentes lignes dans de nombreux threads.
- Moins de changements pour les annulations.
- Permet de verrouiller une seule ligne pendant une longue période.

Inconvénients du verrouillage au niveau de la ligne:

- Prend plus de mémoire que les verrous au niveau de la page ou de la table.
- Est plus lent que les verrous au niveau de la page ou de la table lorsqu'il est utilisé sur une grande partie de la table car vous devez acquérir beaucoup plus de verrous.

Avec les verrous de niveau supérieur, vous pouvez également prendre en charge plus facilement des verrous de différents types pour optimiser l'application, car la charge de verrouillage est inférieure à celle des verrous de niveau ligne.

Les verrous de table sont supérieurs aux verrous de page ou de ligne dans les cas suivants:

- La plupart des instructions pour la table sont des lectures.

Table Lock

Read Lock

Un verrou READ présente les caractéristiques suivantes:

- Un verrou READ pour une table peut être acquis par plusieurs sessions simultanément. De plus, d'autres sessions peuvent lire des données de la table sans acquérir le verrou.
- La session qui détient le verrou READ peut uniquement lire les données de la table, mais ne peut pas écrire. De plus, les autres sessions ne peuvent pas écrire de données dans la table tant que le verrou READ n'est pas libéré. Les opérations d'écriture d'une autre session seront mises dans les états en attente jusqu'à ce que le verrou READ soit libéré.
- Si la session est terminée, normalement ou de manière anormale, MySQL libérera tous les verrous implicitement. Cette fonctionnalité est également pertinente pour le verrou WRITE.

Connection 1	Connection 2

`mysql> create table t (i integer);`
`Query OK, 0 rows affected (0.08 sec)`
 	
`mysql> lock table t read;`
`Query OK, 0 rows affected (0.00 sec)`
 	
`mysql> insert into t values(3);`
`ERROR 1099 (HY000): Table 't' was locked`
`with a READ lock and can't be updated`
	`mysql> insert into t values(3);`
	`-- la connexion est bloqué`

Write Lock

Un verrou WRITE présente les caractéristiques suivantes:

- La seule session qui détient le verrou d'une table peut lire et écrire des données à partir de la table.
- D'autres sessions ne peuvent pas lire et écrire des données dans la table tant que le verrou WRITE n'est pas libéré.

Connection 1	Connection 2

`mysql> lock table testtransaction write;`
`Query OK, 0 rows affected (0,00 sec)`
 	
`mysql> insert into testtransaction values(1,2);`
`Query OK, 1 row affected (0,00 sec)`
	`select * from testtransaction;`
	`-- la connection 2 est bloqué`

Statut des lock

Il est possible d'avoir le statut des connection via SHOW PROCESS LIST

```
*************************** 4. row ***************************
     Id: 10
   User: root
   Host: localhost:33736
     db: transaction
Command: Query
   Time: 28
  State: Waiting for table metadata lock
   Info: select * from testtransaction
4 rows in set (0,00 sec)
```

Kill

Il est parfois nécéssaire de tuer une transaction afin de libérer les lock.
Dans ce cas, on doit récuperer la liste des process et executer une requete KILL

```
SHOW PROCESSLIST \G;
*************************** 1. row ***************************
```

```
     Id: 4
   User: event_scheduler
   Host: localhost
     db: NULL
Command: Daemon
   Time: 16478
  State: Waiting on empty queue
   Info: NULL
*************************** 2. row ***************************
     Id: 8
   User: root
   Host: localhost:33716
     db: myisamtest
Command: Sleep
   Time: 5193
  State:
   Info: NULL
*************************** 3. row ***************************
     Id: 9
   User: root
   Host: localhost:33734
     db: transaction
Command: Sleep
   Time: 5
  State:
   Info: NULL
*************************** 4. row ***************************
     Id: 11
   User: root
   Host: localhost:33738
     db: transaction
Command: Query
   Time: 0
  State: starting
   Info: SHOW PROCESSLIST
4 rows in set (0,00 sec)

ERROR:
No query specified
```

```
mysql> kill 9
    -> ;
Query OK, 0 rows affected (0,00 sec)
```

KILL autorise un modificateur CONNECTION ou QUERY:

- KILL CONNECTION est identique à KILL sans modificateur: il met fin à la connexion associée à processlist_id donné, après avoir terminé toute instruction en cours d'exécution par la connexion.
- KILL QUERY met fin à l'instruction en cours d'exécution de la connexion, mais la laisse elle-même intacte.

Si vous avez le privilège PROCESSUS, vous pouvez voir tous les threads. Si vous disposez du privilège CONNECTION_ADMIN ou SUPER, vous pouvez supprimer tous les threads et toutes les instructions. Sinon, vous ne pouvez voir et tuer que vos propres discussions et déclarations.

Row Lock

Si vous interrogez des données puis insérez ou mettez à jour des données associées dans la même transaction, l'instruction SELECT standard ne donne pas une protection suffisante. D'autres transactions peuvent mettre à jour ou supprimer les mêmes lignes que celles que vous venez de consulter. InnoDB prend en charge deux types de lectures de verrouillage offrant une sécurité supplémentaire:

SELECT... FOR SHARE

Définit un verrou de mode partagé sur toutes les lignes lues. D'autres sessions peuvent lire les lignes, mais ne peuvent pas les modifier jusqu'à ce que votre transaction soit validée. Si l'une de ces lignes a été modifiée par une autre transaction qui n'a pas encore été validée, votre requête attend jusqu'à la fin de cette transaction, puis utilise les dernières valeurs.

Connection 1	Connection 2

```
mysql> begin ;
Query OK, 0 rows affected (0,00 sec)

mysql> select * from testtransaction where
i=2 FOR SHARE;
+------+------------+
| i    | testcolumn |
+------+------------+
|    2 |       NULL |
+------+------------+
1 row in set (0,00 sec)



mysql> insert into testtransaction
values(1,2);
Query OK, 1 row affected (0,00 sec)
```

```
 mysql> begin;
Query OK, 0 rows affected (0,02 sec)

mysql> update testtransaction set
testcolumn='pilou';
-- la transaction est bloqué
ERROR 1205 (HY000): Lock wait timeout
exceeded;
try restarting transaction
```

seule les lignes impacté par le select sont locké

Connection 1	Connection 2

```
mysql> select * from testtransaction where
i=2 FOR SHARE;
+------+------------+
| i    | testcolumn |
+------+------------+
|    2 |       NULL |
+------+------------+
1 row in set (0,00 sec)
```

```
mysql> begin;
Query OK, 0 rows affected (0,02 sec)

mysql> update testtransaction set
testcolumn=45 where i=1;
Query OK, 2 rows affected (0,00 sec)
Rows matched: 2  Changed: 2  Warnings: 0
```

SELECT ... FOR UPDATE

Une commande SELECT ... FOR UPDATE va lire les dernières données disponibles pour chaque ligne, et pose un verrou dessus en même tant qu'il lit. De cette façon, il pose le même verrou que la commande UPDATE .

Connection 1	Connection 2

```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from SolrCoresPreallocated
order by id limit 1 for update;
+----+-------------+-----+-----+
| id | used_status | sid | cid |
+----+-------------+-----+-----+
|  1 |           0 |   0 | 400 |
+----+-------------+-----+-----+
1 row in set (0.00 sec)
```

```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from SolrCoresPreallocated
order by id limit 1 for update;
ERROR 1205 (HY000): Lock wait timeout
exceeded; try restarting transaction
mysql>  rollback;
Query OK, 0 rows affected (0.00 sec)
```

NOWAIT

Si une ligne est verrouillée par une transaction, une transaction SELECT ... FOR UPDATE ou SELECT ... FOR SHARE demandant la même ligne verrouillée doit attendre que la transaction bloquante libère le verrou de ligne. Ce comportement empêche les transactions de mettre à jour ou de supprimer les lignes qui sont interrogées pour des mises à jour par d'autres transactions. Cependant, il n'est pas nécessaire d'attendre qu'un verrou de ligne soit libéré si vous souhaitez que la requête soit renvoyée immédiatement lorsqu'une ligne demandée est verrouillée, ou si le fait d'exclure des lignes verrouillées du jeu de résultats est acceptable.

Pour éviter d'attendre que d'autres transactions libèrent des verrous de ligne, les options NOWAIT et SKIP LOCKED peuvent être utilisées avec les instructions de lecture SELECT ... FOR UPDATE ou SELECT ... FOR SHARE.

NOWAIT

Une lecture de verrouillage qui utilise NOWAIT n’attend jamais d’acquérir un verrou de ligne. La requête s'exécute immédiatement, échouant avec une erreur si une ligne demandée est verrouillée.

SKIP LOCKED

Une lecture de verrouillage qui utilise SKIP LOCKED n’attend jamais d’obtenir un verrou de ligne. La requête s'exécute immédiatement, en supprimant les lignes verrouillées du jeu de résultats.

Administration d'un base de donnée

Création de Base de données

CREATE DATABASE où comment créer une nouvelle base de données dans MySQL Server.
Pour créer une nouvelle base de données dans MySQL, utilisez l'instruction CREATE DATABASE avec la syntaxe suivante:
<source lang='bash'>

```
CREATE DATABASE [IF NOT EXISTS] nom_base[CHARACTER SET charset][COLLATE collation]
```

Tout d'abord, vous spécifiez le nom de la base de données après la clause CREATE DATABASE. Le nom de la base de données doit être unique dans l'instance du serveur MySQL. Si vous essayez de créer une base de données avec un nom qui existe déjà, MySQL génère une erreur.

```
mysql> create database testdb;
Query OK, 1 row affected (0,08 sec)

mysql> create database testdb;
ERROR 1007 (HY000): Can't create database 'testdb'; database exists
mysql>
```

Deuxièmement, pour éviter une erreur en cas de création accidentelle d'une base de données existante, vous pouvez spécifier l'option IF NOT EXISTS. Dans ce cas, MySQL ne génère pas d'erreur, mais termine l'instruction CREATE DATABASE.

Troisièmement, vous pouvez spécifier le jeu de caractères et le classement de la nouvelle base de données au moment de la création. Si vous omettez les clauses CHARACTER SET et COLLATE, MySQL utilise le jeu de caractères et le classement par défaut de la nouvelle base de données.

```
mysql> create database testUtf8 CHARACTER SET utf8 COLLATE utf8_unicode_ci;
Query OK, 1 row affected, 2 warnings (0,05 sec)
```

Le jeu de caractères et la collation de la base de données affectent les aspects suivants du fonctionnement du serveur:

- Pour les instructions CREATE TABLE, le jeu de caractères de la base de données et le classement sont utilisés comme valeurs par défaut pour les définitions de table si le jeu de caractères de la table et le classement ne sont pas spécifiés.
- Pour les instructions LOAD DATA qui n'incluent pas de clause CHARACTER SET, le serveur utilise le jeu de caractères indiqué par la variable système character_set_database pour interpréter les informations contenues dans le fichier.
- Pour les routines stockées (procédures et fonctions), le jeu de caractères de la base de données et le classement en vigueur au moment de la création de la routine sont utilisés comme jeu de caractères et le classement des paramètres de données de caractères pour lesquels la déclaration ne comprend aucun CHARACTER SET ni aucun attribut COLLATE. Pour remplacer cela, indiquez explicitement CHARACTER SET et COLLATE.

```
mysql> SHOW CHARACTER SET ;
+----------+---------------------------------+--------------------+--------+
| Charset | Description | Default collation | Maxlen |
+----------+---------------------------------+--------------------+--------+
| armscii8 | ARMSCII-8 Armenian | armscii8_general_ci | 1 |
| ascii | US ASCII | ascii_general_ci | 1 |
| big5 | Big5 Traditional Chinese | big5_chinese_ci | 2 |
| binary | Binary pseudo charset | binary | 1 |
| cp1250 | Windows Central European | cp1250_general_ci | 1 |
| cp1251 | Windows Cyrillic | cp1251_general_ci | 1 |
| cp1256 | Windows Arabic | cp1256_general_ci | 1 |
| cp1257 | Windows Baltic | cp1257_general_ci | 1 |
| cp850 | DOS West European | cp850_general_ci | 1 |
| cp852 | DOS Central European | cp852_general_ci | 1 |
| cp866 | DOS Russian | cp866_general_ci | 1 |
| cp932 | SJIS for Windows Japanese | cp932_japanese_ci | 2 |
| dec8 | DEC West European | dec8_swedish_ci | 1 |
| eucjpms | UJIS for Windows Japanese | eucjpms_japanese_ci | 3 |
| euckr | EUC-KR Korean | euckr_korean_ci | 2 |
| gb18030 | China National Standard GB18030 | gb18030_chinese_ci | 4 |
| gb2312 | GB2312 Simplified Chinese | gb2312_chinese_ci | 2 |
| gbk | GBK Simplified Chinese | gbk_chinese_ci | 2 |
| geostd8 | GEOSTD8 Georgian | geostd8_general_ci | 1 |
| greek | ISO 8859-7 Greek | greek_general_ci | 1 |
| hebrew | ISO 8859-8 Hebrew | hebrew_general_ci | 1 |
| hp8 | HP West European | hp8_english_ci | 1 |
| keybcs2 | DOS Kamenicky Czech-Slovak | keybcs2_general_ci | 1 |
```

```
| koi8r | KOI8-R Relcom Russian | koi8r_general_ci | 1 |
| koi8u | KOI8-U Ukrainian | koi8u_general_ci | 1 |
| latin1 | cp1252 West European | latin1_swedish_ci | 1 |
| latin2 | ISO 8859-2 Central European | latin2_general_ci | 1 |
| latin5 | ISO 8859-9 Turkish | latin5_turkish_ci | 1 |
| latin7 | ISO 8859-13 Baltic | latin7_general_ci | 1 |
| macce | Mac Central European | macce_general_ci | 1 |
| macroman | Mac West European | macroman_general_ci | 1 |
| sjis | Shift-JIS Japanese | sjis_japanese_ci | 2 |
| swe7 | 7bit Swedish | swe7_swedish_ci | 1 |
| tis620 | TIS620 Thai | tis620_thai_ci | 1 |
| ucs2 | UCS-2 Unicode | ucs2_general_ci | 2 |
| ujis | EUC-JP Japanese | ujis_japanese_ci | 3 |
| utf16 | UTF-16 Unicode | utf16_general_ci | 4 |
| utf16le | UTF-16LE Unicode | utf16le_general_ci | 4 |
| utf32 | UTF-32 Unicode | utf32_general_ci | 4 |
| utf8 | UTF-8 Unicode | utf8_general_ci | 3 |
| utf8mb4 | UTF-8 Unicode | utf8mb4_0900_ai_ci | 4 |
+----------+---------------------------------+--------------------+--------+
41 rows in set (0,00 sec)
```

Il est possible de savoir pour une base de donnée le jeux de données et la collation utilisé:

```
mysql> SELECT @@character_set_database, @@collation_database;
+--------------------------+----------------------+
| @@character_set_database | @@collation_database |
+--------------------------+----------------------+
| utf8mb4 | utf8mb4_0900_ai_ci |
+--------------------------+----------------------+
1 row in set (0,00 sec)
```

```
show databases;
+--------------------+
| Database |
+--------------------+
| information_schema |
| mysql |
| performance_schema |
```

```
| sys |
| testUtf8 |
| testdb |
+--------------------+
6 rows in set (0,00 sec)
```

Enfin, pour accéder à la base de données nouvellement créée, vous utilisez la commande USE database comme suit:

```
mysql> USE testdb;
```

Utilisation d'une base de données

Si USE n'a jamais été utilisé, la requete suivante renverras NULL

```
select database();
+------------+
| database() |
+------------+
| NULL |
+------------+
1 row in set (0,00 sec)
```

Après l'utilisation de USE

```
mysql> use testdb;
Database changed
mysql> select database();
+------------+
| database() |
+------------+
| testdb |
+------------+
```

```
1 row in set (0,00 sec)
```

Storage Engine

```
show engines;
+--------------------+---------
+------------------------------------------------------------+--------------+------
+------------+
| Engine | Support | Comment | Transactions | XA | Savepoints |
+--------------------+---------
+------------------------------------------------------------+--------------+------
+------------+
| FEDERATED | NO | Federated MySQL storage engine | NULL | NULL | NULL |
| InnoDB | DEFAULT | Supports transactions, row-level locking, and foreign keys | YES | YES |
YES |
| PERFORMANCE_SCHEMA | YES | Performance Schema | NO | NO | NO |
| MyISAM | YES | MyISAM storage engine | NO | NO | NO |
| MRG_MYISAM | YES | Collection of identical MyISAM tables | NO | NO | NO |
| BLACKHOLE | YES | /dev/null storage engine (anything you write to it disappears) | NO | NO |
NO |
| MEMORY | YES | Hash based, stored in memory, useful for temporary tables | NO | NO | NO |
| CSV | YES | CSV storage engine | NO | NO | NO |
| ARCHIVE | YES | Archive storage engine | NO | NO | NO |
+--------------------+---------
+------------------------------------------------------------+--------------+------
+------------+
9 rows in set (0,00 sec)
```

Lors de la création de table, il est possible d'utiliser un des storage engine (ici CSV)

```
CREATE TABLE t2 (i INT NOT NULL) ENGINE = CSV;
Query OK, 0 rows affected (0,08 sec)

mysql> insert into t2 values(2);
```

```
Query OK, 1 row affected (0,03 sec)
```

Dans le datadir de mysql, on va trouver un repertoire correspondant à la base de données ainsi que le fichier de metadata et de données de la table en CSV:

```
pilou@lubuntu:~/mysql80/mysql-8.0.13-linux-glibc2.12-x86_64/data/testdb$ ls
t2_336.sdi  t2.CSM  t2.CSV
pilou@lubuntu:~/mysql80/mysql-8.0.13-linux-glibc2.12-x86_64/data/testdb$ more t2.CSV
2
```

En plus de stocker les métadonnées relatives aux objets de base de données dans le dictionnaire de données, MySQL 8 les stocke sous forme sérialisée. Ces données sont appelées informations de dictionnaire sérialisées (SDI). InnoDB stocke les données SDI dans ses fichiers de tablespace. D'autres moteurs de stockage stockent les données SDI dans des fichiers .sdi créés dans le répertoire du schéma. Les données SDI sont générées dans un format JSON compact.

Il est possible de positionner une table vers une autre storage engine:

```
alter table t2 engine INNODB;
Query OK, 1 row affected (0,17 sec)
Records: 1 Duplicates: 0 Warnings: 0
```

En général, les storage engines se doivent de vérifier la transformation des tables:

```
mysql> create table t3 (i INT) ;
Query OK, 0 rows affected (0,10 sec)

mysql> alter table t3 engine CSV;
ERROR 1178 (42000): The storage engine for the table doesn't support nullable columns
```

Renommer une table

Il est possible de renommer une table via la commande RENAME : RENAME ancien_nom TO nouveau_nom

```
rename table t3 to t4;
Query OK, 0 rows affected (0,14 sec)
```

Mais le renommage doit être fait en faisant attention au dépendance

```
create table idtable ( id INT auto_increment, PRIMARY KEY (id));
Query OK, 0 rows affected (0,11 sec)

mysql> insert into idtable values(0);
Query OK, 1 row affected (0,09 sec)

mysql> create view testview as select * from idtable;
Query OK, 0 rows affected (0,09 sec)

mysql> rename table idtable to myidtables;
Query OK, 0 rows affected (0,11 sec)

mysql> select * from testview;
ERROR 1356 (HY000): View 'testdb.testview' references invalid table(s) or column(s) or
function(s) or definer/invoker of view lack rights to use them
```

Administration des tables

Optimiser les tables MySQL et défragmenter pour récupérer de l'espace

Si votre application effectue beaucoup de suppressions et de mises à jour sur la base de données MySQL, il est très probable que vos fichiers de données MySQL soient fragmentés.

Cela entraînera beaucoup d'espace inutilisé et pourrait également affecter les performances.

Il est donc fortement recommandé de défragmenter vos tables MySQL de manière continue.

Mise en place de la table de test

Il est interessant de se doter d'une table de test

```
create table randomint(value integer);
INSERT INTO randomint ( value ) VALUES ( rand() * 3333 );


-- repeter l'opration suivante plusieurs fois


INSERT INTO randomint ( value ) SELECT value * rand() FROM randomint;
```

Requete utiles

La requête suivante permet de lister toutes les bases de données et de calculer la taille de chacune d’elles en Mo.

```
SELECT table_schema AS NomBaseDeDonnees, ROUND(SUM( data_length + index_length ) / 1024 /
1024, 2) AS BaseDonneesMo
FROM information_schema.TABLES GROUP BY TABLE_SCHEMA;
```

Il est également possible de visualiser en détail la taille d'une base de données spécifiquement. La requête ci-dessous présente la taille d'une base en Mo, en Ko et aussi en octets. Il faut juste adapter la requête en remplaçant "nom_base_de_donnees" par la base de votre choix.

```
SELECT CONCAT( SUM(ROUND( ( (DATA_LENGTH + INDEX_LENGTH - DATA_FREE) / 1024 / 1024),2)), 'Mo'
) AS TailleMo,
CONCAT( SUM(ROUND( ( (DATA_LENGTH + INDEX_LENGTH - DATA_FREE) / 1024 ),2)), 'Ko' ) AS
TailleKo,
CONCAT( SUM(ROUND( ( (DATA_LENGTH + INDEX_LENGTH - DATA_FREE) ),2)), 'o' ) AS Tailleo
FROM information_schema.TABLES
WHERE TABLE_SCHEMA = 'nom_base_de_donnees';
```

Pour connaître la taille de chaque table inclus dans une base il est possible d'exécuter la requête ci-dessous :

```
SELECT TABLE_NAME,
CONCAT(ROUND(((DATA_LENGTH + INDEX_LENGTH - DATA_FREE) / 1024 / 1024), 2), 'Mo') AS TailleMo
FROM information_schema.TABLES
WHERE TABLE_SCHEMA = 'nom_base_de_donnees'
```

Pour connaître uniquement la taille d'une table SQL il est possible d'exécuter la requête suivante :

```
SELECT
CONCAT(ROUND(((DATA_LENGTH + INDEX_LENGTH - DATA_FREE) / 1024 / 1024), 2), 'Mo') AS TailleMo
FROM information_schema.TABLES
WHERE TABLE_SCHEMA = 'nom_base_de_donnees'
AND   TABLE_NAME = 'nom_table';
```

data_length et data_free

l'instruction suivante permet d'afficher la data_lenght et data_free

```
mysql> select table_name,data_length,data_free from information_schema.tables where table_name
like 'randomint';
+------------+-------------+-----------+
| TABLE_NAME | DATA_LENGTH | DATA_FREE |
+------------+-------------+-----------+
|  randomint |   1589248   |   4194304 |
+------------+-------------+-----------+

1 row in set (0,00 sec)
```

ou alors:

```
mysql> show table status \G
*************************** 1. row ***************************
Name: randomint
Engine: InnoDB
Version: 10
Row_format: Dynamic
Rows: 32960
Avg_row_length: 48
Data_length: 1589248
Max_data_length: 0
Index_length: 0
Data_free: 2097152
Auto_increment: NULL
Create_time: 2019-01-14 11:34:49
Update_time: NULL
Check_time: NULL
Collation: utf8mb4_0900_ai_ci
Checksum: NULL
Create_options:
Comment:
1 row in set (0,00 sec)
```

DATA_LENGTH

Pour MyISAM, DATA_LENGTH est la longueur du fichier de données, en octets.

Pour InnoDB, DATA_LENGTH est la quantité approximative de mémoire allouée pour l'index clusterisé, en octets. Plus précisément, il s’agit de la taille de l’index clusterisé, exprimée en pages, multipliée par la taille de la page InnoDB.

MAX_DATA_LENGTH

Pour MyISAM, MAX_DATA_LENGTH est la longueur maximale du fichier de données. Il s'agit du nombre total d'octets de données pouvant être stockés dans la table, en fonction de la taille du pointeur de données utilisé.

Non utilisé pour InnoDB.

INDEX_LENGTH

Pour MyISAM, INDEX_LENGTH est la longueur du fichier d'index, en octets.

Pour InnoDB, INDEX_LENGTH est la quantité approximative de mémoire allouée pour les index non clusterisés, en octets. Plus précisément, il s'agit de la somme des tailles d'index non clusterisées, en pages, multipliée par la taille de page InnoDB.

DATA_FREE

Nombre d'octets alloués mais non utilisés.

Les tables InnoDB indiquent l'espace libre de l'espace de table auquel la table appartient. Pour une table située dans l'espace de table partagé, il s'agit de l'espace libre de l'espace de table partagé. Si vous utilisez plusieurs espaces de table et que la table possède son propre espace de table, l'espace disponible ne concerne que cette table. Espace libre signifie le nombre d'octets dans des étendues complètement libres moins une marge de sécurité. Même si l'espace libre affiche 0, il est possible d'insérer des lignes tant qu'il n'est pas nécessaire d'attribuer de nouvelles extensions.

DATA_FREE indique l’espace alloué sur le disque pour un tableau ou un fragment de données de disque, mais n’est pas utilisé par ce dernier. (L'utilisation des ressources de données en mémoire est signalée par la colonne DATA_LENGTH.)
La taille du fichier est

```
-rw-r----- 1 pilou pilou 7340032 janv.  14 11:34 randomint.ibd
```

Le Data_free est difficile à prédire.

- Si la table a été créée avec innodb_file_per_table = 0, il s'agit de l'espace disponible dans ibdata1.
- Si la table n'est pas partitionnée et qu'elle est minuscule, elle est généralement nulle.
- Non partitionné et d’une taille d’au moins mégaoctets, Data_free fait habituellement exactement 4 Mo, 5 Mo, 6 Mo ou 7 Mo. En effet, InnoDB obtient une "extension" de 8 Mo en anticipant le besoin de davantage d’espace.

Pour les tables partitioné(avant les "partitions natives" de 5.7.xx), chaque PARTITION ressemble à une table et se comporte comme une table. Ainsi, le Data_free pour la table est vraiment la somme des 4-7 Mo pour chaque partition.
La table INFORMATION_SCHEMA.TABLES contient environ 20 colonnes, mais pour déterminer la quantité d’espace disque utilisée par les tables, nous nous intéresserons plus particulièrement à deux colonnes: DATA_LENGTH et INDEX_LENGTH.

- DATA_LENGTH est la longueur (ou la taille) de toutes les données de la table (en octets).
- INDEX_LENGTH est la longueur (ou la taille) du fichier d'index de la table (également en octets).

Armés de ces informations, nous pouvons exécuter une requête qui listera toutes les tables d’une base de données spécifique ainsi que l’espace disque (taille) de chacune. Nous pouvons même devenir un peu plus sophistiqués et convertir les valeurs de taille normale d'octets en quelque chose de plus utile et compréhensible pour la plupart des gens, comme les mégaoctets.

```
mysql> SELECT TABLE_NAME AS 'Table', ROUND((DATA_LENGTH + INDEX_LENGTH) / 1024 / 1024) AS
'Size (MB)'
 FROM information_schema.TABLES WHERE TABLE_name LIKE "randomint" ORDER BY (DATA_LENGTH +
INDEX_LENGTH) ;

+-----------+-----------+
|    Table  | Size (MB) |
+-----------+-----------+
| randomint |         2 |
+-----------+-----------+

1 row in set (0,00 sec)
```

ANALYZE TABLE

L’optimiseur de requête MySQL est un composant important du serveur MySQL qui crée un plan d’exécution de requête optimal pour une requête. Pour une requête particulière, l'optimiseur de requête utilise la distribution de clé stockée et d'autres facteurs pour décider de l'ordre dans lequel les tables doivent être jointes lors de l'exécution de la jointure et de l'index à utiliser pour une table spécifique.

Cependant, les distributions de clés peuvent parfois être inexactes, par exemple, après avoir effectué de nombreuses modifications de données dans la table, notamment insérer, supprimer ou mettre à jour. Si la distribution de clé n'est pas exacte, l'optimiseur de requête peut choisir un plan d'exécution de requête incorrect pouvant entraîner un problème de performances majeur.

```
mysql> analyze table randomint;

+---------------------+---------+----------+----------+
|                Table |      Op |  Msg_type |  Msg_text |
+---------------------+---------+----------+----------+
| myisamtest.randomint |  analyze |   status |       OK |
+---------------------+---------+----------+----------+

1 row in set (0,03 sec)
```

OPTIMIZE TABLE

Lorsque vous travaillez avec la base de données, vous effectuez de nombreuses modifications, telles que l'insertion, la mise à jour et la suppression de données dans la table, susceptibles de fragmenter le stockage physique de la table. En conséquence, les performances du serveur de base de données sont dégradées.

MySQL vous fournit une instruction qui vous permet d'optimiser la table pour éviter ce problème de défragmentation.

```
mysql> optimize table randomint;

+---------------------+----------+----------
+-------------------------------------------------------------------+
|  Table               |       Op |  Msg_type
|                                                          Msg_text |
```

```
+----------------------+----------+----------
+-------------------------------------------------------------------+
| myisamtest.randomint | optimize | note     | Table does not support optimize, doing recreate
+ analyze instead |
| myisamtest.randomint | optimize | status
|                                                                   OK |
+----------------------+----------+----------
+-------------------------------------------------------------------+

2 rows in set (0,88 sec)
```

InnoDB ne supporte pas OPTIMIZE comme MyISAM. Il crée une table vide et y copie toutes les lignes de la table existante. Il supprime essentiellement l'ancienne table, la renomme, puis exécute un ANALYZE pour collecter des statistiques. C'est ce qui se rapproche le plus d'InnoDB pour réaliser une OPTIMISATION.

Supprimons les données

```
mysql> delete from randomint;

Query OK, 32768 rows affected (0,88 sec)

mysql> show table status \G

*************************** 1. row ***************************
Name: randomint
Engine: InnoDB
Version: 10
Row_format: Dynamic
Rows: 32960
Avg_row_length: 48
Data_length: 1589248
Max_data_length: 0
Index_length: 0
Data_free: 2097152
Auto_increment: NULL
Create_time: 2019-01-14 11:34:49
Update_time: NULL
Check_time: NULL
Collation: utf8mb4_0900_ai_ci
```

```
Checksum: NULL
Create_options:
Comment:

1 row in set (0,00 sec)
```

Ce qui ne change pas grand chose dans MySQL, mais sur le disque

```
-rw-r----- 1 pilou pilou 114688 janv. 14 12:00 randomint.ibd
```

Pour raffraichir les statistiques, il faut faire un analyze table

```
mysql> analyze table randomint;

+---------------------+---------+----------+----------+
| Table               | Op      | Msg_type | Msg_text |
+---------------------+---------+----------+----------+
| myisamtest.randomint | analyze | status   |       OK |
+---------------------+---------+----------+----------+

1 row in set (0,09 sec)

□
mysql> show table status \G

*************************** 1. row ***************************
Name: randomint
Engine: InnoDB
Version: 10
Row_format: Dynamic
Rows: 0
Avg_row_length: 0
Data_length: 16384
Max_data_length: 0
Index_length: 0
Data_free: 0
Auto_increment: NULL
Create_time: 2019-01-14 12:00:11
```

```
Update_time: NULL
Check_time: NULL
Collation: utf8mb4_0900_ai_ci
Checksum: NULL
Create_options:
Comment:

1 row in set (0,00 sec)
```

Format des tables

Afin de voir les différents format, nous allons créer une table de champs TEXT

```
mysql> create table compacttable (c1 LONGTEXT) row_format=compact;
Query OK, 0 rows affected (0,06 sec)

mysql> insert into compacttable values (REPEAT("0123456789", 10000));
Query OK, 1 row affected (0,10 sec)

mysql> select length(REPEAT("0123456789", 10000));
+-----------------------------------+
| length(REPEAT("0123456789", 10000)) |
+-----------------------------------+
|                            100000 |
+-----------------------------------+

1 row in set (0,00 sec)
```

Statut des tables

Il est possible d'avoir l'état d'une table ainsi

```
mysql> SHOW TABLE STATUS IN test1\G

*************************** 1. row ***************************
           Name: t1
         Engine: InnoDB
        Version: 10
     Row_format: Dynamic
           Rows: 0
 Avg_row_length: 0
    Data_length: 16384
Max_data_length: 0
  Index_length: 16384
      Data_free: 0
 Auto_increment: 1
    Create_time: 2016-09-14 16:29:38
    Update_time: NULL
     Check_time: NULL
      Collation: latin1_swedish_ci
       Checksum: NULL
 Create_options:
        Comment:
```

Erreur possible

Le format de ligne par défaut pour la table InnoDB est défini par la variable innodb_default_row_format. La valeur par défaut est différente dans MySQL 5.6 et 5.7. Si vous êtes maintenant conscient de cette différence et que vous utilisez la valeur par défaut de MySQL 5.6, vous pouvez rencontrer des problèmes tels que:

Row Format	`REDUNDANT`	`COMPACT`	`DYNAMIC`	`COMPRESSED`
Compact Storage Characteristics	No	Yes	Yes	Yes
Enhanced Variable-Length Column Storage	No	No	Yes	Yes
Large Index Key Prefix Support	No	No	Yes	Yes
Compression Support	No	No	No	Yes
Supported Tablespace Types	system, file-per table, general*	system, file-per-table, general*	file-per-table, general*	file-per-table, general*

Row Format	`REDUNDANT`	`COMPACT`	`DYNAMIC`	`COMPRESSED`
Required File Format	Antelope or Barracuda	Antelope or Barracuda	Barracuda	Barracuda

Compact

Compact est le format par défaut et est généralement approprié pour le format de fichier Antelope. Introduit dans MySQL 5.0.

Dans ce format (comme dans Redundant) les colonnes BLOB et TEXT sont partiellement stockées dans les pages de données. Au moins 767 octets sont stockés dans la ligne; tous les débords sont stockés dans des pages dédiées. Comme les tailles de lignes de Compact et Redundant sont d'environ 8000 octets, cela limite le nombre de colonnes BLOB ou TEXT qui peuvent être utilisées dans une table. Chaque page de BLOB page contient 16Ko, sans compter les données.

Les autres colonnes peuvent être stockées dans des pages différentes si elles dépassent la taille de ligne maximum par page.

Redundant

Redundant est l'ancien, format non compressé supporté par les anciennes versions de MySQL. C'était le seul format dispobible avant la version 5.0 et a été le mode par défaut dans MySQL 5.0.3. Il est recommandé de paramétrer innodb_strict_mode lorsque vous l'utilisez.

Dynamic

Les tables en format Dynamic contiennent des enregistrements de taille variable, ce qui permet de mieux exploiter l'espace disque que Compact ou Redundant, notamment pour les tables contenant des BLOBs, mais moins que le format Compressed.

Il ne peut être utilisé qu'avec le format de fichier XtraDB Barracuda, et nécessite que les tables et index soient stockés dans leur propre tablespace, ce qui implique que les variables systèmes soient parametrées innodb_file_per_table=1 et innodb_file_format=barracuda. Il est recommandé de parametrer innodb_strict_mode lors de l'utilisation de ce format.

La taille maximum des lignes est désormais de 65535 octets.

Avec le format Dynamic (et Compressed), les colonnes BLOB et TEXT sont stockées différemment de Compact. Si les données ne peuvent être contenues dans une ligne de page, alors seulement un pointeur sera stocké et contiendra l'adresse de la page dédiée. Chaque page externe contient une partie des données et l'adresse de la page suivante, si nécessaire. Les pointeurs ont une taille de 20Ko. Cela permet de stocker un grand nombre de colonnes BLOB ou de TEXT dans une table.

Compressed

Le format Compressed permet de réduire la taille des données. Il ne peut être utilisé qu'avec le format de fichier XtraDB Barracuda, et nécessite que les tables et index soient stockés dans leur propre tablespace, ce qui implique que les variables systèmes soient parametrées innodb_file_per_table=1 et innodb_file_format=barracuda. Il est recommandé de parametrer innodb_strict_mode lors de l'utilisation de ce format.

Le fait d'utiliser le format Compressed reduit aussi le KEY_BLOCK_SIZE par défaut. Si KEY_BLOCK_SIZE est omis de la clause CREATE TABLE ou ALTER TABLE, il sera par défaut à 8Ko - traditionnellement c'est 16Ko (cf. innodb_page_size). Il est aussi possible de régler le KEY_BLOCK_SIZE à 1Ko, 2Ko, 4Ko ou 16Ko. Le fait de le régler à 16Ko, la taille standard, provoquera généralement une compression minimale à moins qu'il n'y ait beaucoup de colonnes de type BLOB, TEXT ou VARCHAR.

Notez que préciser un KEY_BLOCK_SIZE spécifique dans une définition de table provoquera automatiquement une compression - Il n'est donc pas nécessaire de préciser l'option ROW_FORMAT=COMPRESSED.

Afin d'évider trop de compression/décompression de pages, XtraDB/InnoDB essaye de garder les pages compressées et decompressées dans le buffer pool, quand il y a assez d'espace. Cela donne un cache plus gros. Lorsque la place vient à manquer, un algorithme de LRU vient décider de la suppression de pages compressées ou decompressées du buffer: pour soulager les CPUs, les pages compressées sont supprimées en priorité; pour soulager les I/O, les pages non compressées sont supprimées en priorité. Bien entendu, quand cela est nécessaire, la règle s'applique aux pages compressées et non compressées.