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.

Revision #8
Created 26 November 2019 19:24:29 by Admin
Updated 9 June 2020 14:35:54 by ggpilou2