Version: 2023/11/20 12:51

SER407 - Optimisation

Matériel

Processeur

La vitesse d'un processeur est la caractéristique la plus importante. Il n'est pas possible de paralléliser une requête sur plusieurs processeurs. Pour cette raison, une requête occupe un processeur à la fois. Prenons le cas d'un serveur esclave dans le contexte d'une réplication. Toutes les requêtes de réplication passeront par un seul processeur, rendant les autres processeurs éventuels inutiles.

Mémoire

La mémoire est un facteur critique pour le serveur MariaDB. La donnée la plus importante est la quantité de données consultées. Prenons le cas d'une base de données de 20Go où seules les données de la dernière semaine sont consultées. Si ces données ne représentent que 1Go, un serveur muni de 2Go suffira largement.

Disque Dur

Les performances d'un disque dur se mesurent avec deux paramètres, le temps d'accès et le débit. Avec une application transactionnelle de boutique Internet, le temps d'accès est le facteur le plus important. Si par contre, l'application est faite pour stocker l'ensemble des livres dans une bibliothèque, le débit devient très important lors des requêtes de recherche pour trouver un livre spécifique.

Système d'Exploitation

MariaDB est disponible pour Windows™, Solaris™ et Linux. Le choix du Système d'Exploitation est une question de coût, de compétences internes et de politique de l'infrastructure informatique.

Cache de Requêtes

Le cache des requêtes est utilisé par MariaDB lors des requêtes de type SELECT. Le cache est utile dans le cas où la majorité des requêtes sont de type lecture. Lors d'un SELECT, MariaDB vérifie si la requête est déjà dans le cache. Cette vérification est faite pour **toutes** les requêtes, mêmes celles qui sont **impropres**. Il ne faut donc **pas** activer le cache quand :

- les écritures sont nombreuses,
- les lectures ne sont pas répétées,
- le requêtes sont en grande partie **impropre**.

L'activation du cache se fait en modifiant la valeur de l'option **query_cache_type** :

```
MariaDB [(none)]> SHOW VARIABLES LIKE 'query_cache_type';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| query_cache_type | ON    |
+------------------+-------+
1 row in set (0.00 sec)

MariaDB [(none)]>
```

Cette option peut prendre trois valeurs :

- **ON** : le serveur essaie de mettre en cache toutes les requêtes SELECT sauf celles comportant la phrase SQL_NO_CACHE,
- **OFF** : aucune requête n'est mise en cache,

- **DEMAND** : le serveur essaie de mettre en cache toutes les requêtes SELECT comportant la phrase SQL_CACHE.

Exclusions

Seules les requêtes dites déterministes peuvent être mises en cache. Par exemple, une requête de type SELECT * FROM table donnera toujours le même résultat tant que la table n'est pas modifiée. Cette requête peut être mise dans le cache. Par contre la requête SELECT CURRENT_TIMESTAMP() est non-déterministe. Par conséquent, la requête est dite **impropre**.

Requêtes

MariaDB recherche dans le cache chaque fois qu'il y a une requête SELECT. MariaDB est sensible à la case. Pour cette raison, MariaDB considère que les trois requêtes suivantes sont différentes :

- SELECT prenom,nom FROM familles,
- select prenom,nom FROM familles,
- SELECT nom,prenom FROM familles.

Invalidations

Le serveur MariaDB met à jour automatiquement le contenu du cache. Lors d'une modification d'une table au niveau de sa structure ou bien au niveau de ses données, le serveur MariaDB cherche l'efficacité et supprime du cache **toutes** les requêtes impliquant la table concernée.

Effacements

Dans le cas où le cache n'est pas suffisant pour stocker une requête, MariaDB procède a un effacement de l'entrée la plus ancienne. Ce processus s’appelle un **effacement**.

Fragmentation

MariaDB divise le cache en blocs mémoire d'une taille égale. Quand une requête est mise en cache, le serveur MariaDB ne connaît pas à l'avance le nombre d'enregistrements qui vont être retournées. De ce fait, il ne connaît pas non plus l'occupation de la mémoire. Si la requête n'occupe pas un nombre entier de blocs, la mémoire concernée est perdue. Dans ce cas on parle de **fragmentation**.

Paramètres

Les options associées avec le cache sont :

Option	**Descripton**
query_cache_size	La taille de la mémoire dédiée au cache. La mémoire est affectée par multiples de 1 024 octets. Dans la pratique il ne faut pas dépasser les 256 Mo.
query_cache_limit	La taille maximale d'une requête qui peut être mise en cache
query_cache_type	Indique le type de cache
query_cache_min_res_unit	La taille des blocs de mémoire du cache
query_cache_wlock_invalidate	Quand cette option est **OFF**, le serveur peut renvoyer le contenu du cache même si la table est verrouillée par un autre utilisateur

Verification du Cache

Pour consulter la configuration du cache, utilisez la requête suivante :

```
MariaDB [(none)]> SHOW GLOBAL STATUS LIKE 'Qcache%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Qcache_free_blocks      | 0     |
| Qcache_free_memory      | 0     |
| Qcache_hits             | 0     |
| Qcache_inserts          | 0     |
```

```
| Qcache_lowmem_prunes    | 0     |
| Qcache_not_cached       | 0     |
| Qcache_queries_in_cache | 0     |
| Qcache_total_blocks     | 0     |
+-------------------------+-------+
8 rows in set (0.01 sec)
```

Cette configuration s'explique de la façon suivante :

Option	**Descripton**
Qcache_free_memory	La taille de la mémoire cache libre
Qcache_lowmem_prunes	Le nombre d'entrées supprimées dans le cache par défaut d'assez d'espace disque
Qcache_hits	Le nombre de fois que le serveur a pu servir une requête du cache

Les valeurs ci-dessus doivent être comparées avec le nombre total de requêtes de type SELECT :

```
MariaDB [(none)]> SHOW GLOBAL STATUS LIKE 'Com_select';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Com_select    | 14    |
+---------------+-------+
1 row in set (0.00 sec)
```

L'efficacité du cache peut être calculé en utilisant le formule suivant :

```
100 * Qcache_hits / Com_select
```

Le taux d'insertion dans le cache est le résultat du formule suivant :

```
100 * Qcache_inserts / Com_select
```

La fragmentation peut être evaluée par le formule :

```
Qcache_free_blocks / Qcache_total_blocks
```

> **Important** : Pour défragmenter le cache, il convient d'utiliser la commande **FLUSH QUERY CACHE**. Il est à noter que cette commande ne vide pas le contenu du cache. Pour vider le contenu du cache, il convient d'utiliser la commande **RESET QUERY CACHE**.

Optimisation du Schéma

L'optimisation du schéma est un aspect primordial de la bonne gestion de MariaDB. Les règles de base sont les suivantes :

- Plus le type de données est petit et compact, plus il sera léger et performant,
- Évitez des colonnes NULL car cela demande plus de travail au serveur que des colonnes NOT NULL,
- Utilisez des champs adéquats aux besoins réels de la donnée à stocker,
- Ne surdimensionnez pas les champs de texte car MariaDB utilise la taille maximale pour effectuer des tris. Privilégiez donc des champs VARCHAR par rapport CHAR car ce dernier a une longueur fixe.

PROCEDURE ANALYSE

Afin de vous aider, MariaDB fournit une commande capable d'examiner les données réellement présentes dans une table et de vous conseiller sur le type de champ à adopter :

```
MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| CarnetAdresses     |
| ligue1             |
```

```
| mysql              |
| performance_schema |
| test               |
+--------------------+
6 rows in set (0.01 sec)

MariaDB [(none)]> USE ligue1;
Database changed
MariaDB [(none)]> SHOW TABLES;
+-----------------+
| Tables_in_ligue1 |
+-----------------+
| equipe          |
| rencontre       |
+-----------------+
2 rows in set (0.00 sec)

MariaDB [(none)]> SHOW CREATE TABLE equipe\G
*************************** 1. row ***************************
       Table: equipe
Create Table: CREATE TABLE `equipe` (
  `id_equipe` int(11) NOT NULL AUTO_INCREMENT,
  `nom` varchar(50) NOT NULL,
  `stade` varchar(50) NOT NULL,
  `ville` varchar(30) NOT NULL,
  `points` int(11) NOT NULL DEFAULT '0',
  `buts` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`id_equipe`)
) ENGINE=MyISAM AUTO_INCREMENT=4 DEFAULT CHARSET=latin1
1 row in set (0.00 sec)

MariaDB [(none)]> SELECT * FROM equipe PROCEDURE ANALYSE()\G
*************************** 1. row ***************************
             Field_name: ligue1.equipe.id_equipe
```

```
              Min_value: 1
              Max_value: 3
             Min_length: 1
             Max_length: 1
       Empties_or_zeros: 0
                  Nulls: 0
Avg_value_or_avg_length: 2.0000
                    Std: 0.8165
      Optimal_fieldtype: ENUM('1','2','3') NOT NULL
*************************** 2. row ***************************
             Field_name: ligue1.equipe.nom
              Min_value: Debian AC
              Max_value: Vista FC
             Min_length: 8
             Max_length: 11
       Empties_or_zeros: 0
                  Nulls: 0
Avg_value_or_avg_length: 9.3333
                    Std: NULL
      Optimal_fieldtype: ENUM('Debian AC','FC Mandriva','Vista FC') NOT NULL
*************************** 3. row ***************************
             Field_name: ligue1.equipe.stade
              Min_value: Parc des Princes
              Max_value: Yankee Stadium
             Min_length: 11
             Max_length: 16
       Empties_or_zeros: 0
                  Nulls: 0
Avg_value_or_avg_length: 13.6667
                    Std: NULL
      Optimal_fieldtype: ENUM('Parc des Princes','Qwest Field','Yankee Stadium') NOT NULL
*************************** 4. row ***************************
             Field_name: ligue1.equipe.ville
              Min_value: New York
```

```
              Max_value: Redmond
             Min_length: 5
             Max_length: 8
       Empties_or_zeros: 0
                  Nulls: 0
Avg_value_or_avg_length: 6.6667
                    Std: NULL
      Optimal_fieldtype: ENUM('New York','Paris','Redmond') NOT NULL
*************************** 5. row ***************************
             Field_name: ligue1.equipe.points
              Min_value: 0
              Max_value: 0
             Min_length: 1
             Max_length: 1
       Empties_or_zeros: 3
                  Nulls: 0
Avg_value_or_avg_length: 0.0000
                    Std: 0.0000
      Optimal_fieldtype: ENUM('0') NOT NULL
*************************** 6. row ***************************
             Field_name: ligue1.equipe.buts
              Min_value: 0
              Max_value: 0
             Min_length: 1
             Max_length: 1
       Empties_or_zeros: 3
                  Nulls: 0
Avg_value_or_avg_length: 0.0000
                    Std: 0.0000
      Optimal_fieldtype: ENUM('0') NOT NULL
6 rows in set (0.00 sec)

MariaDB [(none)]>
```

Normalisation

La normalisation d'une base de données consiste en la structuration des objets de façon à obtenir un modèle de données performant et sur.

Les trois règles les plus importantes sont :

- Eviter des colonnes qui se répètent,
- Eviter l'incohérence des données,
- Tous les champs doivent dépendre uniquement de la clef primaire.

LAB #1 - Normalisation

Créez la base de données **connaissances** :

```
MariaDB [(none)]> CREATE DATABASE connaissances;
Query OK, 1 row affected (0.00 sec)
MariaDB [(none)]> USE connaissances;
Database changed
```

Utilisez la requête suivante pour créer la table **employe** :

```
CREATE TABLE employe (
id int(11) NOT NULL,
nom varchar(30) NOT NULL DEFAULT '',
ville varchar(30) NOT NULL DEFAULT '',
savoir1 varchar(30) NOT NULL DEFAULT '',
niv1 tinyint(4) NOT NULL,
savoir2 varchar(30) NOT NULL DEFAULT '',
niv2 tinyint(4) NOT NULL,
PRIMARY KEY (id)
) ENGINE=InnoDB;
```

```
MariaDB [(none)]> CREATE TABLE employe (
    -> id int(11) NOT NULL,
    -> nom varchar(30) NOT NULL DEFAULT '',
    -> ville varchar(30) NOT NULL DEFAULT '',
    -> savoir1 varchar(30) NOT NULL DEFAULT '',
    -> niv1 tinyint(4) NOT NULL,
    -> savoir2 varchar(30) NOT NULL DEFAULT '',
    -> niv2 tinyint(4) NOT NULL,
    -> PRIMARY KEY (id)
    -> ) ENGINE=InnoDB;
Query OK, 0 rows affected (0.06 sec)

MariaDB [(none)]>
```

Utilisez la requête suivante pour injecter des données :

```
INSERT INTO employe (id , nom , ville , savoir1 , niv1 , savoir2 , niv2) VALUES ('1' , 'Alex', 'Londres',
'Linux', '10','Solaris', '8');INSERT INTO employe (id , nom , ville , savoir1 , niv1 , savoir2 , niv2) VALUES
('2' , 'Mathieu', 'Paris', 'Apache', '5','MariaDB', '7');INSERT INTO employe (id, nom , ville , savoir1 , niv1 ,
savoir2 , niv2) VALUES ('3' , 'Thomas', 'Paris', 'Linux', '5','MariaDB', '7');
```

```
MariaDB [(none)]> INSERT INTO employe (id , nom , ville , savoir1 , niv1 , savoir2 , niv2) VALUES ('1' , 'Alex',
'Londres', 'Linux', '10','Solaris', '8');INSERT INTO employe (id , nom , ville , savoir1 , niv1 , savoir2 , niv2)
VALUES ('2' , 'Mathieu', 'Paris', 'Apache', '5','MariaDB', '7');INSERT INTO employe (id, nom , ville , savoir1 ,
niv1 , savoir2 , niv2) VALUES ('3' , 'Thomas', 'Paris', 'Linux', '5','MariaDB', '7');
Query OK, 1 row affected (0.00 sec)

Query OK, 1 row affected (0.01 sec)

Query OK, 1 row affected (0.00 sec)
```

Vous devez obtenir le résultat suivant :

```
MariaDB [(none)]> SELECT * FROM employe;
+----+---------+---------+---------+------+---------+------+
| id | nom     | ville   | savoir1 | niv1 | savoir2 | niv2 |
+----+---------+---------+---------+------+---------+------+
|  1 | Alex    | Londres | Linux   |   10 | Solaris |    8 |
|  2 | Mathieu | Paris   | Apache  |    5 | MariaDB |    7 |
|  3 | Thomas  | Paris   | Linux   |    5 | MariaDB |    7 |
+----+---------+---------+---------+------+---------+------+
3 rows in set (0.01 sec)

MariaDB [(none)]>
```

Indexes

Un index est une structure de données liée à une table dont le rôle est de permettre le serveur de trouver une référence au plus vite.

Un index peut être créé à la création de la table avec la clause KEY ou INDEX ou bien après sa création avec une des commandes suivantes :

```
CREATE INDEX nom_index ON table (colonne);
```

```
ALTER TABLE table ADD INDEX (colonne);
```

Ces deux commandes on des différences :

- Avec CREATE INDEX le nom de l'index est obligatoire,
- Avec ALTER TABLE si le nom de l'index n'est pas spécifié, MariaDB en crée un,
- Avec ALTER TABLE il est possible de créer plusieurs index en même temps en séparant les clauses ADD INDEX par des virgules.

Pour supprimer un index on utilise une des deux commandes suivantes :

```
DROP INDEX nom_index
```

```
ALTER TABLE nom_index
```

Types d'Index

Index Uniques

Un index unique comporte une contrainte. Toutes les valeurs non NULL doivent être unique. Par contre il est possible d'avoir plusieurs valeurs NULL :

```
MariaDB [(none)]> use indexes;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed

MariaDB [(none)]> INSERT INTO t (id) VALUES (1);
Query OK, 1 row affected (0.01 sec)

MariaDB [(none)]> INSERT INTO t (id) VALUES (1);
ERROR 1062 (23000): Duplicate entry '1' for key 'id'

MariaDB [(none)]> INSERT INTO t (id) VALUES (NULL);
Query OK, 1 row affected (0.00 sec)

MariaDB [(none)]> INSERT INTO t (id) VALUES (NULL);
Query OK, 1 row affected (0.02 sec)

MariaDB [(none)]> SELECT * FROM t;
+------+
| id   |
+------+
| NULL |
| NULL |
```

```
|    1 |
+------+
3 rows in set (0.00 sec)

MariaDB [(none)]>
```

Clef Primaires

Une Clef Primaire ressemble à un index unique mais possède deux différences :

- Il ne peut y avoir qu'une Clef Primaire par table,
- Une Clef Primaire ne peut pas contenir des valeurs NULL.

```
MariaDB [(none)]> CREATE TABLE t1 (id INT);
Query OK, 0 rows affected (0.03 sec)

MariaDB [(none)]> CREATE INDEX `PRIMARY` ON t1;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MariaDB
server version for the right syntax to use near '' at line 1

MariaDB [(none)]> ALTER TABLE t1 ADD PRIMARY KEY (ID);
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Index sur Plusieurs Colonnes

```
MariaDB [(none)]> CREATE TABLE t2 (id INT, col1 VARCHAR(30), col2 VARCHAR(30));
Query OK, 0 rows affected (0.33 sec)

MariaDB [(none)]> CREATE INDEX index_col12 ON t2 (col1,col2);
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
MariaDB [(none)]>
```

Important : MariaDB est capable d'exploiter un préfixe à gauche de l'index multicolonne. Cela implique que si MariaDB juge que seul **col1** est nécessaire pour faire le travail, il est capable de l'isoler de l'index multicolonne.

Index sur un Préfixe de Colonne

```
MariaDB [(none)]> CREATE TABLE t3 (col TEXT);
Query OK, 0 rows affected (0.94 sec)

MariaDB [(none)]> CREATE INDEX index_prefixe ON t3 (col);
ERROR 1170 (42000): BLOB/TEXT column 'col' used in key specification without a key length

MariaDB [(none)]> CREATE INDEX index_prefixe ON t3 (col(50));
Query OK, 0 rows affected (0.10 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [(none)]>
```

Important : Pour les index stockant du texte, il est possible de ne créer l'index que sur les N premiers caractères du champ.

Clefs Etrangères

Une clef étrangère indique à MariaDB de rejeter tout ajout ou modification dans la table enfant si la valeur de la clef n'a pas de correspondance dans la table parent.

```
MariaDB [(none)]> CREATE TABLE t4 (
    -> id INT(11) NOT NULL AUTO_INCREMENT,
    -> nom VARCHAR(20) NOT NULL DEFAULT'',
    -> PRIMARY KEY (id),
    -> KEY index_nom (nom)
    -> ) ENGINE = InnoDB;
Query OK, 0 rows affected (0.32 sec)

MariaDB [(none)]> CREATE TABLE t5 ( id INT(11) NOT NULL AUTO_INCREMENT, ref_nom VARCHAR(20) NOT NULL DEFAULT'',
PRIMARY KEY (id), CONSTRAINT foreign_index FOREIGN KEY foreign_ref_nom (ref_nom) REFERENCES t4(nom)) ENGINE =
InnoDB;
Query OK, 0 rows affected (0.31 sec)

MariaDB [(none)]>
```

Important : La clause CONSTRAINT est optionnelle. Le nom de la contrainte doit être unique dans toute la base. Le nom de l'index est optionnel. La clef étrangère peut référencer plusieurs champs en même temps mais tous les champs doivent être dans la même table.

Lors d'une suppression ou d'une mise à jour, le comportement est dicté par l'ajout d'une clause supplémentaire dans la clause FOREIGN KEY :

- **RESTRICT** ou **NO ACTION** - C'est l'action par défaut. Toute tentative de mise à jour ou de suppression dans la table parente provoque une erreur si une ou plusieurs lignes correspondantes se trouvent dans la table enfant,
- **CASCADE** - Quand une ligne est mise à jour ou supprimée de la table parente, la ou les lignes corrspondantes dans la table enfant sont mises à jour ou supprimées,
- **SET NULL** - Quand une ligne est mise à jour ou supprimée de la table parente, la ou les lignes corrspondantes dans la table enfant sont mises à NULL.

Index Cluster avec InnoDB

Le moteur InnoDB stocke les données **avec** la clef primaire. Par conséquent toute requête qui utilise la clef primaire est très efficace. Par contre toute requête qui utilise une clef secondaire est moins efficace car MariaDB recherche d'abord dans l'index secondaire la clef primaire associée puis recherche les données à partir de la clef primaire.

> **Important** : Dans le cas où une clef primaire n'existe pas, MariaDB essaie de trouver une clef unique ne contentant aucune valeur NULL. S'il n'en trouve pas, il crée une clef primaire cachée.

Index Couvrant

Dans le cas où toutes les données nécessaires à la résolution d'une requête se trouvent dans un index, on parle d'un index **couvrant**. Dans ce cas, l'optimiseur de MariaDB n'a pas besoin de chercher les données.

> **Important** : Un index peut être créé avec un de deux algorithmes principaux : B-Tree ou Hash. Un index couvrant est forcément de type B-Tree.

Index FULLTEXT

Dans le cas où il est nécessaire de rechercher une chaîne de caractères dans un champs, on doit utiliser un index FULLTEXT :

```
MariaDB [(none)]> CREATE TABLE t6(
    -> col1 VARCHAR(50),
    -> col2 VARCHAR(50),
```

```
    -> col3 VARCHAR(50),
    -> FULLTEXT index_col1_col2 (col1,col2)
    -> );
ERROR 1214 (HY000): The used table type doesn't support FULLTEXT indexes
```

Important : Le moteur InnoDB supporte les index FULLTEXT à partir de la version 5.6 de MariaDB.

```
MariaDB [(none)]> CREATE TABLE t6( col1 VARCHAR(50), col2 VARCHAR(50), col3 VARCHAR(50), FULLTEXT index_col1_col2
(col1,col2) ) ENGINE=MyISAM;
Query OK, 0 rows affected (0.04 sec)

MariaDB [(none)]> CREATE FULLTEXT INDEX index_col2 ON t6 (col2);
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [(none)]> ALTER TABLE t6 ADD FULLTEXT index_col3 (col3);
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [(none)]>
```

Téléchargez la base de donnée exemple sakila :

```
[root@centos ~]# wget http://downloads.mysql.com/docs/sakila-db.tar.gz
--2014-02-01 11:46:10--  http://downloads.mysql.com/docs/sakila-db.tar.gz
Résolution de downloads.mysql.com... 137.254.60.14
Connexion vers downloads.mysql.com|137.254.60.14|:80...connecté.
requête HTTP transmise, en attente de la réponse...200 OK
Longueur: 722633 (706K) [application/x-gzip]
Sauvegarde en : «sakila-db.tar.gz»
```

```
100%[======================================>] 722 633     20,2K/s   ds 14s

2014-02-01 11:46:30 (50,6 KB/s) - «sakila-db.tar.gz» sauvegardé [722633/722633]
```

Editez le fichier ~/sakila-db/sakila-schema.sql avec VI en exécutant la commande **:g/ENGINE=InnoDB/s//ENGINE=MyISAM/g** :

```
[root@centos7 ~]# vi ~/sakila-db/sakila-schema.sql
```

Installez sakila :

```
[root@centos ~]# tar xvf sakila-db.tar.gz
sakila-db/
sakila-db/sakila-schema.sql
sakila-db/sakila.mwb
sakila-db/sakila-data.sql
[root@centos ~]# mysql -uroot -p < sakila-db/sakila-schema.sql
Enter password:
[root@centos ~]# mysql -uroot -p < sakila-db/sakila-data.sql
Enter password:
```

Mode Langage Naturel

L'appel à une recherche FULLTEXT nécessite l'utilisation de clauses spécifiques, à savoir **MATCH()** et **AGAINST()** :

```
MariaDB [(none)]> USE sakila;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [(none)]> SELECT film_id,
    -> LEFT(description,30),
    -> MATCH(title,description)
```

```
    -> AGAINST('Database Administrator') AS pertinence
    -> FROM film_text
    -> WHERE MATCH(title,description)
    -> AGAINST('Database Administrator')
    -> LIMIT 5;
+---------+---------------------------------+-------------------+
| film_id | LEFT(description,30)            | pertinence        |
+---------+---------------------------------+-------------------+
|     113 | A Thrilling Yarn of a Database | 6.662790775299072 |
|     962 | A Fanciful Story of a Database | 6.662790775299072 |
|      27 | A Amazing Reflection of a Data | 6.662790775299072 |
|     363 | A Fast-Paced Display of a Car  | 4.527374267578125 |
|     372 | A Taut Display of a Cat And a  | 4.527374267578125 |
+---------+---------------------------------+-------------------+
5 rows in set (0.00 sec)

MariaDB [(none)]>
```

Important : Par défaut une requête FULLTEXT est une recherches en langage naturel. Ceci implique que chaque mot de la clause AGAINST est recherché dans l'index et que la pertinence est calculé en fonction de la fréquences d'apparition. Notez que si un mot est peu fréquent dans l'ensemble des lignes indexées et ce mot fait partie de ceux recherchés, la pertinence de la ligne trouvée sera plus haute que la pertinence de la ligne contenant un mot peu qui est fréquent dans les lignes. Dernièrement et par défaut les mots faisant partie de au moins 50% des enregistrements sont automatiquement ignorés. Il n'est pas possible de modifier ce pourcentage.

Mode Booléen

En utilisant le mode booléen, il vous est possible d'ajouter des **modificateurs** pour préciser quels mots ont une pertinence plus élevée que d'autres :

Modificateur	Description
+	Le résultat doit contenir ce mot
-	Le résultat ne doit pas contenir ce mot
~	La pertinence de ce mot est plus faible que les autres mots
	La pertinence de ce mot est plus haute que les autres mots
*	Placé à la fin d'une chaine, ce modificateur est un joker

Par exemple :

```
MariaDB [(none)]> SELECT film_id,title,description FROM film_text WHERE MATCH(title,description) AGAINST('Data*
Administrator +Anaconda' IN BOOLEAN MODE)\G*************************** 1. row ***************************
    film_id: 23
      title: ANACONDA CONFESSIONS
description: A Lacklusture Display of a Dentist And a Dentist who must Fight a Girl in Australia
*************************** 2. row ***************************
    film_id: 315
      title: FINDING ANACONDA
description: A Fateful Tale of a Database Administrator And a Girl who must Battle a Squirrel in New Orleans
2 rows in set (0.01 sec)

MariaDB [(none)]>
```

Important : Notez que seulement deux résultats sont retournés. En effet, il n'y a que deux enregistrements dans la base qui contiennent le mot Anaconda. Notez que les mots faisant partie de au moins 50% des enregistrements **ne** sont **pas** automatiquement ignorés.

Mode Expansion de Requête

Dans ce mode, la recherche est lancée deux fois. La première fois avec les mots de la clause AGAINST, la deuxième avec les mots ainsi que les résultats les plus pertinents.

```
MariaDB [(none)]> SELECT film_id,title,description FROM film_text WHERE MATCH(title,description)
AGAINST('MariaDB');
+---------+-------------------------+-----------------------------------------------------------------------------
----------------------------------------+
| film_id | title                   | description
|
+---------+-------------------------+-----------------------------------------------------------------------------
----------------------------------------+
|     428 | HOMICIDE PEACH          | A Astounding Documentary of a Hunter And a Boy who must Confront a Boy in A
MariaDB Convention                        |
|     577 | MILE MULAN              | A Lacklusture Epistle of a Cat And a Husband who must Confront a Boy in A
MariaDB Convention                         |
|     812 | SMOKING BARBARELLA      | A Lacklusture Saga of a Mad Cow And a Mad Scientist who must Sink a Cat in A
MariaDB Convention                        |
|     936 | VANISHING ROCKY         | A Brilliant Reflection of a Man And a Woman who must Conquer a Pioneer in A
MariaDB Convention                         |
|     494 | KARATE MOON             | A Astounding Yarn of a Womanizer And a Dog who must Reach a Waitress in A
MariaDB Convention                         |
|     608 | MURDER ANTITRUST        | A Brilliant Yarn of a Car And a Database Administrator who must Escape a Boy
in A MariaDB Convention                   |
|     341 | FROST HEAD              | A Amazing Reflection of a Lumberjack And a Cat who must Discover a Husband
in A MariaDB Convention                    |
|     773 | SEABISCUIT PUNK         | A Insightful Saga of a Man And a Forensic Psychologist who must Discover a
Mad Cow in A MariaDB Convention            |
|     242 | DOOM DANCING            | A Astounding Panorama of a Car And a Mad Scientist who must Battle a
Lumberjack in A MariaDB Convention                |
|     819 | SONG HEDWIG             | A Amazing Documentary of a Man And a Husband who must Confront a Squirrel in
A MariaDB Convention                      |
|      15 | ALIEN CENTER            | A Brilliant Drama of a Cat And a Mad Scientist who must Battle a Feminist in
A MariaDB Convention                      |
|     137 | CHARADE DUFFEL          | A Action-Packed Display of a Man And a Waitress who must Build a Dog in A
```

```
MariaDB Convention                                |
|     844 | STEERS ARMAGEDDON      | A Stunning Character Study of a Car And a Girl who must Succumb a Car in A
MariaDB Convention                                |
|     119 | CAPER MOTIONS          | A Fateful Saga of a Moose And a Car who must Pursue a Woman in A MariaDB
Convention                                        |
|     907 | TRANSLATION SUMMER     | A Touching Reflection of a Man And a Monkey who must Pursue a Womanizer in A
MariaDB Convention                                |
|     899 | TOWERS HURRICANE       | A Fateful Display of a Monkey And a Car who must Sink a Husband in A MariaDB
Convention                                        |
|     918 | TWISTED PIRATES        | A Touching Display of a Frisbee And a Boat who must Kill a Girl in A MariaDB
Convention                                        |
|     933 | VAMPIRE WHALE          | A Epic Story of a Lumberjack And a Monkey who must Confront a Pioneer in A
MariaDB Convention                                |
|     716 | REAP UNFAITHFUL        | A Thrilling Epistle of a Composer And a Sumo Wrestler who must Challenge a
Mad Cow in A MariaDB Convention                   |
|     727 | RESURRECTION SILVERADO | A Epic Yarn of a Robot And a Explorer who must Challenge a Girl in A MariaDB
Convention                                        |
|     809 | SLIPPER FIDELITY       | A Taut Reflection of a Secret Agent And a Man who must Redeem a Explorer in
A MariaDB Convention                              |
|     567 | MEET CHOCOLATE         | A Boring Documentary of a Dentist And a Butler who must Confront a Monkey in
A MariaDB Convention                              |
|     129 | CAUSE DATE             | A Taut Tale of a Explorer And a Pastry Chef who must Conquer a Hunter in A
MariaDB Convention                                |
|     261 | DUFFEL APOCALYPSE      | A Emotional Display of a Boat And a Explorer who must Challenge a Madman in
A MariaDB Convention                              |
|     213 | DATE SPEED             | A Touching Saga of a Composer And a Moose who must Discover a Dentist in A
MariaDB Convention                                |
|      11 | ALAMO VIDEOTAPE        | A Boring Epistle of a Butler And a Cat who must Fight a Pastry Chef in A
MariaDB Convention                                |
|     201 | CYCLONE FAMILY         | A Lacklusture Drama of a Student And a Monkey who must Sink a Womanizer in A
MariaDB Convention                                |
|     352 | GATHERING CALENDAR     | A Intrepid Tale of a Pioneer And a Moose who must Conquer a Frisbee in A
MariaDB Convention                                |
|     398 | HANOVER GALAXY         | A Stunning Reflection of a Girl And a Secret Agent who must Succumb a Boy in
```

```
A MariaDB Convention                          |
|     426 | HOME PITY              | A Touching Panorama of a Man And a Secret Agent who must Challenge a Teacher
in A MariaDB Convention                       |
|      72 | BILL OTHERS            | A Stunning Saga of a Mad Scientist And a Forensic Psychologist who must
Challenge a Squirrel in A MariaDB Convention   |
|     845 | STEPMOM DREAM          | A Touching Epistle of a Crocodile And a Teacher who must Build a Forensic
Psychologist in A MariaDB Convention          |
|     870 | SWARM GOLD             | A Insightful Panorama of a Crocodile And a Boat who must Conquer a Sumo
Wrestler in A MariaDB Convention               |
|     466 | INTOLERABLE INTENTIONS | A Awe-Inspiring Story of a Monkey And a Pastry Chef who must Succumb a
Womanizer in A MariaDB Convention              |
|     974 | WILD APOLLO            | A Beautiful Story of a Monkey And a Sumo Wrestler who must Conquer a A Shark
in A MariaDB Convention                       |
|     980 | WIZARD COLDBLOODED     | A Lacklusture Display of a Robot And a Girl who must Defeat a Sumo Wrestler
in A MariaDB Convention                       |
|     551 | MAIDEN HOME            | A Lacklusture Saga of a Moose And a Teacher who must Kill a Forensic
Psychologist in A MariaDB Convention          |
|     987 | WORDS HUNTER           | A Action-Packed Reflection of a Composer And a Mad Scientist who must Face a
Pioneer in A MariaDB Convention               |
|     183 | CONVERSATION DOWNHILL  | A Taut Character Study of a Husband And a Waitress who must Sink a Squirrel
in A MariaDB Convention                       |
|     804 | SLEEPING SUSPECTS      | A Stunning Reflection of a Sumo Wrestler And a Explorer who must Sink a
Frisbee in A MariaDB Convention                |
|     576 | MIGHTY LUCK            | A Astounding Epistle of a Mad Scientist And a Pioneer who must Escape a
Database Administrator in A MariaDB Convention |
|     733 | RIVER OUTLAW           | A Thrilling Character Study of a Squirrel And a Lumberjack who must Face a
Hunter in A MariaDB Convention                 |
|     303 | FANTASY TROOPERS       | A Touching Saga of a Teacher And a Monkey who must Overcome a Secret Agent
in A MariaDB Convention                       |
|     937 | VARSITY TRIP           | A Action-Packed Character Study of a Astronaut And a Explorer who must Reach
a Monkey in A MariaDB Convention              |
|     782 | SHAKESPEARE SADDLE     | A Fast-Paced Panorama of a Lumberjack And a Database Administrator who must
Defeat a Madman in A MariaDB Convention        |
|     114 | CAMELOT VACATION       | A Touching Character Study of a Woman And a Waitress who must Battle a
```

```
Pastry Chef in A MariaDB Convention             |
|     971 | WHALE BIKINI           | A Intrepid Story of a Pastry Chef And a Database Administrator who must Kill
a Feminist in A MariaDB Convention        |
|     822 | SOUP WISDOM            | A Fast-Paced Display of a Robot And a Butler who must Defeat a Butler in A
MariaDB Convention                          |
|     107 | BUNCH MINDS            | A Emotional Story of a Feminist And a Feminist who must Escape a Pastry Chef
in A MariaDB Convention                     |
|     750 | RUN PACIFIC            | A Touching Tale of a Cat And a Pastry Chef who must Conquer a Pastry Chef in
A MariaDB Convention                        |
+---------+------------------------+-------------------------------------------------------------------------
----------------------------------------+
50 rows in set (0.01 sec)

MariaDB [(none)]>
```

```
MariaDB [(none)]> SELECT film_id,title,description FROM film_text WHERE MATCH(title,description)
AGAINST('MariaDB' WITH QUERY EXPANSION);
...
|     113 | CALIFORNIA BIRDS           | A Thrilling Yarn of a Database Administrator And a Robot who must
Battle a Database Administrator in Ancient India                |
|     617 | NATURAL STOCK              | A Fast-Paced Story of a Sumo Wrestler And a Girl who must Defeat a Car
in A Baloon Factory                                    |
|     424 | HOLOCAUST HIGHBALL         | A Awe-Inspiring Yarn of a Composer And a Man who must Find a Robot in
Soviet Georgia                                          |
|     589 | MODERN DORADO              | A Awe-Inspiring Story of a Butler And a Sumo Wrestler who must Redeem a
Boy in New Orleans                                     |
|     432 | HOPE TOOTSIE               | A Amazing Documentary of a Student And a Sumo Wrestler who must Outgun
a A Shark in A Shark Tank                               |
|     145 | CHISUM BEHAVIOR            | A Epic Documentary of a Sumo Wrestler And a Butler who must Kill a Car
in Ancient India                                       |
|      41 | ARSENIC INDEPENDENCE       | A Fanciful Documentary of a Mad Cow And a Womanizer who must Find a
Dentist in Berlin                                       |
|     751 | RUNAWAY TENENBAUMS         | A Thoughtful Documentary of a Boat And a Man who must Meet a Boat in An
Abandoned Fun House                                    |
```

```
|     962 | WASTELAND DIVINE            | A Fanciful Story of a Database Administrator And a Womanizer who must
Fight a Database Administrator in Ancient China              |
|     177 | CONNECTICUT TRAMP           | A Unbelieveable Drama of a Crocodile And a Mad Cow who must Reach a
Dentist in A Shark Tank                                       |
|     627 | NORTH TEQUILA               | A Beautiful Character Study of a Mad Cow And a Robot who must Reach a
Womanizer in New Orleans                                      |
|     702 | PULP BEVERLY                | A Unbelieveable Display of a Dog And a Crocodile who must Outrace a Man
in Nigeria                                                    |
|     511 | LAWRENCE LOVE               | A Fanciful Yarn of a Database Administrator And a Mad Cow who must
Pursue a Womanizer in Berlin                                   |
|     376 | GRAPES FURY                 | A Boring Yarn of a Mad Cow And a Sumo Wrestler who must Meet a Robot in
Australia                                                     |
|     430 | HOOK CHARIOTS               | A Insightful Story of a Boy And a Dog who must Redeem a Boy in
Australia                                                      |
|     267 | EAGLES PANKY                | A Thoughtful Story of a Car And a Boy who must Find a A Shark in The
Sahara Desert                                                 |
|     968 | WEREWOLF LOLA               | A Fanciful Story of a Man And a Sumo Wrestler who must Outrace a
Student in A Monastery                                         |
|     692 | POTLUCK MIXED               | A Beautiful Story of a Dog And a Technical Writer who must Outgun a
Student in A Baloon                                           |
|     192 | CROSSING DIVORCE            | A Beautiful Documentary of a Dog And a Robot who must Redeem a
Womanizer in Berlin                                            |
|     752 | RUNNER MADIGAN              | A Thoughtful Documentary of a Crocodile And a Robot who must Outrace a
Womanizer in The Outback                                      |
|     448 | IDAHO LOVE                  | A Fast-Paced Drama of a Student And a Crocodile who must Meet a
Database Administrator in The Outback                          |
|     996 | YOUNG LANGUAGE              | A Unbelieveable Yarn of a Boat And a Database Administrator who must
Meet a Boy in The First Manned Space Station                  |
+---------+-----------------------------+-------------------------------------------------------------------------
-------------------------------------------------------------+
999 rows in set (0.02 sec)

MariaDB [(none)]>
```

Configuration

Une recherche FULLTEXT peut être configurée en modifiant les trois variables suivants :

```
MariaDB [(none)]> SHOW VARIABLES LIKE 'ft_min_word_len';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| ft_min_word_len | 4     |
+-----------------+-------+
1 row in set (0.00 sec)

MariaDB [(none)]> SHOW VARIABLES LIKE 'ft_max_word_len';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| ft_max_word_len | 84    |
+-----------------+-------+
1 row in set (0.00 sec)

MariaDB [(none)]> SHOW VARIABLES LIKE 'ft_stopword_list';
Empty set (0.00 sec)

MariaDB [(none)]>
```

Important : La dernière variable peut contenir une liste de mots qui ne seront pas indexés. Notez que chaque fois que vous modifiez une des ces variables, il est nécessaire de reconstruire l'index FULLTEXT avec la commande REPAIR TABLE table QUICK;

Limitations

Veuillez garder en tête les points suivants :

- Pour effectuer une recherche FULLTEXT, il **faut** utiliser MATCH AGAINST,
- Dès que vous utilisez MATCH AGAINST, le serveur utilisera une recherche FULLTEXT même si l'optimiseur trouve un autre index qui aurait été meilleur,
- Les résultats en mode langage naturel sont triés par la pertinence. Ceci n'est **pas** le cas pour les deux autres modes,
- Un index FULLTEXT ne peut **pas** être utilisé comme index couvrant.

La Commande EXPLAIN

La commande EXPLAIN est utilisée pour voir le plan d'exécution de la requête choisi par l'optimiseur :

```
MariaDB [(none)]> EXPLAIN SELECT film_id,title,description FROM film_text WHERE MATCH(title,description)
AGAINST('MariaDB')\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: film_text
         type: fulltext
possible_keys: idx_title_description
          key: idx_title_description
      key_len: 0
          ref:
         rows: 1
        Extra: Using where
1 row in set (0.00 sec)

MariaDB [(none)]>
```

La sortie de la commande démontre 10 colonnes dont les plus importantes sont **type**, **possible_keys**, **keys**, **key_len**, **rows** et **extra** :

Important : Pour l'explication des autres colonnes, voir **ce lien**.

La Colonne type

Cette colonne montre comment l'optimiseur a prévu d'accéder aux données. Les cas suivants peuvent être trouvés en sachant qu'ils sont dans l'ordre du moins performant vers le plus performant :

ALL

Un parcours complet de la table est nécessaire car il n'y a pas d'index disponible ou satisfaisant :

```
MariaDB [(none)]> EXPLAIN SELECT * FROM film_text WHERE title LIKE 'The%'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: film_text
         type: ALL
possible_keys: idx_title_description
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1000
        Extra: Using where
1 row in set (0.00 sec)

MariaDB [(none)]>
```

```
MariaDB [(none)]> SELECT * FROM film_text WHERE title LIKE 'The%'\G
```

```
*************************** 1. row ***************************
    film_id: 886
      title: THEORY MERMAID
description: A Fateful Yarn of a Composer And a Monkey who must Vanquish a Womanizer in The First Manned Space
Station
1 row in set (0.00 sec)

MariaDB [(none)]>
```

index

Un parcours complet d'un index est necéssaire :

```
MariaDB [(none)]> EXPLAIN SELECT category_id FROM category\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: category
         type: index
possible_keys: NULL
          key: PRIMARY
      key_len: 1
          ref: NULL
         rows: 16
        Extra: Using index
1 row in set (0.01 sec)

MariaDB [(none)]>
```

```
MariaDB [(none)]> SELECT category_id FROM category\G
*************************** 1. row ***************************
category_id: 1
*************************** 2. row ***************************
```

```
category_id: 2
*************************** 3. row ***************************
category_id: 3
*************************** 4. row ***************************
category_id: 4
*************************** 5. row ***************************
category_id: 5
*************************** 6. row ***************************
category_id: 6
*************************** 7. row ***************************
category_id: 7
*************************** 8. row ***************************
category_id: 8
*************************** 9. row ***************************
category_id: 9
*************************** 10. row ***************************
category_id: 10
*************************** 11. row ***************************
category_id: 11
*************************** 12. row ***************************
category_id: 12
*************************** 13. row ***************************
category_id: 13
*************************** 14. row ***************************
category_id: 14
*************************** 15. row ***************************
category_id: 15
*************************** 16. row ***************************
category_id: 16
16 rows in set (0.00 sec)

MariaDB [(none)]>
```

range

Un parcours d'une partie d'un index est nécessaire :

```
MariaDB [(none)]> EXPLAIN SELECT * FROM category WHERE category_id > 50\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: category
         type: range
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 1
          ref: NULL
         rows: 1
        Extra: Using where
1 row in set (0.01 sec)

MariaDB [(none)]>
```

```
MariaDB [(none)]> SELECT * FROM category WHERE category_id > 50\G
Empty set (0.01 sec)

MariaDB [(none)]>
```

index_merge

Plusieurs index sont utilisés simultanément :

```
MariaDB [(none)]> EXPLAIN SELECT * FROM rental WHERE rental_id > 10 OR inventory_id < 100\G
*************************** 1. row ***************************
           id: 1
```

```
  select_type: SIMPLE
        table: rental
         type: index_merge
possible_keys: PRIMARY,idx_fk_inventory_id
          key: idx_fk_inventory_id,PRIMARY
      key_len: 3,4
          ref: NULL
         rows: 8557
        Extra: Using sort_union(idx_fk_inventory_id,PRIMARY); Using where
1 row in set (0.01 sec)

MariaDB [(none)]>
```

```
MariaDB [(none)]> SELECT * FROM rental WHERE rental_id > 10 OR inventory_id < 100\G
...
*************************** 13776. row ***************************
   rental_id: 13790
 rental_date: 2005-08-20 12:17:27
inventory_id: 1385
 customer_id: 11
 return_date: 2005-08-25 12:20:27
    staff_id: 1
 last_update: 2006-02-15 21:30:53
*************************** 13777. row ***************************
   rental_id: 13791
 rental_date: 2005-08-20 12:21:05
inventory_id: 1890
 customer_id: 67
 return_date: 2005-08-22 17:58:05
    staff_id: 1
 last_update: 2006-02-15 21:30:53
*************************** 13778. row ***************************
   rental_id: 13792
 rental_date: 2005-08-20 12:21:37
```

```
inventory_id: 4157
 customer_id: 78
 return_date: 2005-08-27 14:28:37
    staff_id: 1
 last_update: 2006-02-15 21:30:53
*************************** 13779. row ***************************
   rental_id: 13793
 rental_date: 2005-08-20 12:22:04
inventory_id: 2598
 customer_id: 424
 return_date: 2005-08-27 09:51:04
    staff_id: 2
 last_update: 2006-02-15 21:30:53
...
```

ref

Un index non unique est examiné. Plusieurs lignes peuvent être retournées :

```
MariaDB [(none)]> EXPLAIN SELECT rental_date FROM rental INNER JOIN customer USING (customer_id) WHERE
customer_id=1\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: customer
         type: const
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 2
          ref: const
         rows: 1
        Extra: Using index
*************************** 2. row ***************************
```

```
           id: 1
  select_type: SIMPLE
        table: rental
         type: ref
possible_keys: idx_fk_customer_id
          key: idx_fk_customer_id
      key_len: 2
          ref: const
         rows: 32
        Extra:
2 rows in set (0.01 sec)

MariaDB [(none)]>
```

```
MariaDB [(none)]> SELECT rental_date FROM rental INNER JOIN customer USING (customer_id) WHERE customer_id=1\G
*************************** 1. row ***************************
rental_date: 2005-05-25 11:30:37
*************************** 2. row ***************************
rental_date: 2005-05-28 10:35:23
*************************** 3. row ***************************
rental_date: 2005-06-15 00:54:12
*************************** 4. row ***************************
rental_date: 2005-06-15 18:02:53
*************************** 5. row ***************************
rental_date: 2005-06-15 21:08:46
*************************** 6. row ***************************
rental_date: 2005-06-16 15:18:57
*************************** 7. row ***************************
rental_date: 2005-06-18 08:41:48
*************************** 8. row ***************************
rental_date: 2005-06-18 13:33:59
*************************** 9. row ***************************
rental_date: 2005-06-21 06:24:45
*************************** 10. row ***************************
```

```
rental_date: 2005-07-08 03:17:05
*************************** 11. row ***************************
rental_date: 2005-07-08 07:33:56
*************************** 12. row ***************************
rental_date: 2005-07-09 13:24:07
*************************** 13. row ***************************
rental_date: 2005-07-09 16:38:01
*************************** 14. row ***************************
rental_date: 2005-07-11 10:13:46
*************************** 15. row ***************************
rental_date: 2005-07-27 11:31:22
*************************** 16. row ***************************
rental_date: 2005-07-28 09:04:45
*************************** 17. row ***************************
rental_date: 2005-07-28 16:18:23
*************************** 18. row ***************************
rental_date: 2005-07-28 17:33:39
*************************** 19. row ***************************
rental_date: 2005-07-28 19:20:07
*************************** 20. row ***************************
rental_date: 2005-07-29 03:58:49
*************************** 21. row ***************************
rental_date: 2005-07-31 02:42:18
*************************** 22. row ***************************
rental_date: 2005-08-01 08:51:04
*************************** 23. row ***************************
rental_date: 2005-08-02 15:36:52
*************************** 24. row ***************************
rental_date: 2005-08-02 18:01:38
*************************** 25. row ***************************
rental_date: 2005-08-17 12:37:54
*************************** 26. row ***************************
rental_date: 2005-08-18 03:57:29
*************************** 27. row ***************************
```

```
rental_date: 2005-08-19 09:55:16
*************************** 28. row ***************************
rental_date: 2005-08-19 13:56:54
*************************** 29. row ***************************
rental_date: 2005-08-21 23:33:57
*************************** 30. row ***************************
rental_date: 2005-08-22 01:27:57
*************************** 31. row ***************************
rental_date: 2005-08-22 19:41:37
*************************** 32. row ***************************
rental_date: 2005-08-22 20:03:46
32 rows in set (0.00 sec)

MariaDB [(none)]>
```

Important : Il existe une variante de ref, appelé ref_or_null. Dans ce cas, il y a une deuxième passe pour retrouver les valeurs NULL.

eq_ref

Un index unique ou une clé primaire est examiné :

```
MariaDB [(none)]> EXPLAIN SELECT email FROM customer INNER JOIN rental USING (customer_id) WHERE rental_date >
'2006-04-10 00:00:00' AND RENTAL_DATE < '2006-05-24 00:00:00'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: rental
         type: range
possible_keys: rental_date,idx_fk_customer_id
```

```
          key: rental_date
      key_len: 8
          ref: NULL
         rows: 1
        Extra: Using where; Using index
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: customer
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 2
          ref: sakila.rental.customer_id
         rows: 1
        Extra:
2 rows in set (0.00 sec)

MariaDB [(none)]>
```

Important : Ce type est le meilleur t(ype d'accès possible.

Cas Spécifiques

const

Une seule ligne a besoin d'être lue dans la table concernée :

```
MariaDB [(none)]> EXPLAIN SELECT * FROM category WHERE category_id=1\G
```

```
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: category
         type: const
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 1
          ref: const
         rows: 1
        Extra:
1 row in set (0.00 sec)

MariaDB [(none)]> SELECT * FROM category WHERE category_id=1\G
*************************** 1. row ***************************
category_id: 1
       name: Action
last_update: 2006-02-15 04:46:27
1 row in set (0.00 sec)

MariaDB [(none)]>
```

System

Identique au type précédent dans le cas où la table est une table à une seule ligne.

NULL

Aucun parcours de table ou d'index n'est nécessaire :

```
MariaDB [(none)]> EXPLAIN SELECT COUNT(*) FROM film_text\G
```

```
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: NULL
         type: NULL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: NULL
        Extra: Select tables optimized away
1 row in set (0.00 sec)

MariaDB [(none)]> SELECT COUNT(*) FROM film_text\G
*************************** 1. row ***************************
COUNT(*): 1000
1 row in set (0.01 sec)

MariaDB [(none)]>
```

Les Colonnes possible_keys, keys et key_len

La colonne **possible_keys** contient l'ensemble des index que l'optimiseur va considérer pour la requête concernée tandis que la colonne **key** indique l'index qui serait utiliser.

Une valeur NULL dans la colonne **possible_keys** indique qu'il n'existe pas d'index utilisable tandis que la valeur NULL dans la colonne **key** indique que l'optimiseur n'a pas trouvé d'index pertinent.

La colonne **key_len** indique la taille en octets de l'index choisi.

Téléchargez la base de données exemple **world** :

```
[root@centos ~]# wget https://downloads.mysql.com/docs/world-db.tar.gz
```

Installez la base de données exemple **world** :

```
[root@centos ~]# mysql -uroot -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 8
Server version: 5.5.33-cll-lve MariaDB Community Server (GPL) by Atomicorp

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

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

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> CREATE DATABASE world;
Query OK, 1 row affected (0.00 sec)

MariaDB [(none)]> exit
Bye
[root@centos ~]# gunzip world-db.tar.gz
[root@centos ~]# mysql -uroot -p world < world-db/world.sql
Enter password:
[root@centos ~]#
```

Voyons un exemple maintenant avec cette base de données :

```
MariaDB [(none)]> USE world;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [(none)]> ALTER TABLE City ADD INDEX idx_name_district(Name,District);
```

```
Query OK, 4079 rows affected (0.11 sec)
Records: 4079  Duplicates: 0  Warnings: 0

MariaDB [(none)]> EXPLAIN SELECT * FROM City WHERE Name LIKE 'San%'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: City
         type: range
possible_keys: idx_name_district
          key: idx_name_district
      key_len: 35
          ref: NULL
         rows: 141
        Extra: Using where
1 row in set (0.01 sec)

MariaDB [(none)]>
```

Important : Notez que la taille de l'index utilisé est de 35 octets.

La Colonne rows

La colonne **rows** contient une estimation du nombre de lignes que le moteur doit parcourir avant de pouvoir retourner un résultat de la requête :

```
MariaDB [(none)]> EXPLAIN SELECT * FROM City CROSS JOIN Country\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: Country
```

```
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 239
        Extra:
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: City
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 4079
        Extra: Using join buffer
2 rows in set (0.00 sec)

MariaDB [(none)]>
```

Important : Dans l'exemple ci-dessus, MariaDB estime avoir besoin de parcourir 239*4079 lignes soit 974 881 enregistrements !!

Pour prouver qu'il s'agit bien d'une estimation, saisissez la requête suivante :

```
MariaDB [(none)]> EXPLAIN SELECT * FROM City WHERE Name LIKE 'San%'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
```

```
        table: City
         type: range
possible_keys: idx_name_district
          key: idx_name_district
      key_len: 35
          ref: NULL
         rows: 141
        Extra: Using where
1 row in set (0.05 sec)

MariaDB [(none)]> SELECT COUNT(*) FROM City WHERE Name LIKE 'San%'\G
*************************** 1. row ***************************
COUNT(*): 110
1 row in set (0.02 sec)

MariaDB [(none)]>
```

Important : Si vous vous rendez compte que les estimations sont largement erronées vous pouvez utilisez la commande ANALYZE TABLE pour mettre à jour les statistiques sur la table. D'une manière générale, plus élevée est la valeur de **rows**, plus lente sera la requête.

La Colonne extra

La colonne extra donne des informations supplémentaires concernant la requête :

- **Using where** - visible quand le serveur n'a pas pu résoudre une clause WHERE en utilisant un index,
- **Using index** - le serveur va utiliser un index couvrant,
- **Using temporary** - le serveur va utiliser une table temporaire pour effectuer un tri,
- **Using filesort** - le serveur doit trier les résultats de la requête.

La Commande EXPLAIN EXTENDED

La commande **EXPLAIN EXTENDED** est disponible dans MariaDB depuis la version 5.1. EXPLAIN EXTENDED donne une approximation, en pourcentage par rapport à la valeur de **rows**, du nombre de lignes qui seront retournées lors de la requête :

```
MariaDB [(none)]> EXPLAIN EXTENDED SELECT ID FROM City WHERE ID < 50\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: City
         type: range
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: NULL
         rows: 50
     filtered: 100.00
        Extra: Using where; Using index
1 row in set, 1 warning (0.00 sec)

MariaDB [(none)]> SELECT COUNT(ID) FROM City WHERE ID < 50\G
*************************** 1. row ***************************
COUNT(ID): 49
1 row in set (0.01 sec)

MariaDB [(none)]>
```

Par contre cette valeur n'est qu'une estimation :

```
MariaDB [(none)]> EXPLAIN EXTENDED SELECT Name FROM City WHERE CountryCode LIKE 'F%' AND district LIKE 'F%' AND
population >100000\G
*************************** 1. row ***************************
           id: 1
```

```
  select_type: SIMPLE
        table: City
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 4079
     filtered: 100.00
        Extra: Using where
1 row in set, 1 warning (0.00 sec)

MariaDB [(none)]> SELECT Name FROM City WHERE CountryCode LIKE 'F%' AND district LIKE 'F%' AND population
>100000\G
*************************** 1. row ***************************
Name: Besançon
1 row in set (0.07 sec)

MariaDB [(none)]>
```

Malgré le peu de fiabilité dans le résultat de la colonne **filtered**, EXPLAIN EXTENDED est une clause bien utile car elle permet de voir la requête réécrite à partir du plan d'exécution :

```
MariaDB [(none)]> EXPLAIN EXTENDED SELECT COUNT(*) FROM Country\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: NULL
         type: NULL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: NULL
```

```
     filtered: NULL
        Extra: Select tables optimized away
1 row in set, 1 warning (0.00 sec)

MariaDB [(none)]> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Note
   Code: 1003
Message: select count(0) AS `COUNT(*)` from `world`.`Country`
1 row in set (0.00 sec)

MariaDB [(none)]>
```

Optimisation des Requêtes

Isolation des Colonnes

L'encapsulation d'une colonne dans une fonction interdit l'utilisation d'un index :

```
MariaDB [(none)]> USE sakila;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [(none)]> EXPLAIN SELECT * FROM rental WHERE TO_DAYS(CURRENT_DATE()) - TO_DAYS(rental_date) < 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: rental
         type: ALL
possible_keys: NULL
```

```
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 15629
        Extra: Using where
1 row in set (0.00 sec)

MariaDB [(none)]> EXPLAIN SELECT * FROM rental WHERE rental_date > CURRENT_DATE() + INTERVAL 10 DAY\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: rental
         type: range
possible_keys: rental_date
          key: rental_date
      key_len: 8
          ref: NULL
         rows: 1
        Extra: Using where
1 row in set (0.00 sec)

MariaDB [(none)]>
```

Important : La réécriture de la requête sans encapsuler une colonne dans une fonction produit une requête plus performante (range > ALL).

Jointures

Quand MariaDB effectue des jointures il le fait une à une et successivement. Par exemple, dans le cas de 4 tables, table1, table2, table3, table4, la table1 est jointe à la table2 produisant une table table12. La table12 est ensite jointe à la table3 produisant la table123. La table123 est ensuite jointe

à la table4 pour produire la table1234.

Dans le cas d'un INNER JOIN, MariaDB peut décider de ne pas respecter cet ordre :

```
MariaDB [(none)]> EXPLAIN SELECT email FROM customer INNER JOIN rental ON customer.customer_id =
rental.customer_id WHERE rental_date > '2006-04-10 00:00:00' AND rental_date < '2006-05-24 00:00:00'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: rental
         type: range
possible_keys: rental_date,idx_fk_customer_id
          key: rental_date
      key_len: 8
          ref: NULL
         rows: 1
        Extra: Using where; Using index
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: customer
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 2
          ref: sakila.rental.customer_id
         rows: 1
        Extra:
2 rows in set (0.01 sec)

MariaDB [(none)]>
```

Important : L'ordre de jointure ici est rental vers customer et non pas customer vers

rental.

Pourquoi l'optimiseur a choisi d'inverser l'ordre de jointure ? Dans l'exemple précédent il faut parcourir 1 ligne. Forcez maintenant la jointure dans l'autre direction :

```
MariaDB [(none)]> EXPLAIN SELECT email FROM customer STRAIGHT_JOIN rental ON customer.customer_id =
rental.customer_id WHERE rental_date > '2006-04-10 00:00:00' AND rental_date < '2006-05-24 00:00:00'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: customer
         type: ALL
possible_keys: PRIMARY
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 577
        Extra:
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: rental
         type: range
possible_keys: rental_date,idx_fk_customer_id
          key: rental_date
      key_len: 8
          ref: NULL
         rows: 1
        Extra: Using where; Using index; Using join buffer
2 rows in set (0.00 sec)

MariaDB [(none)]>
```

Dans le cas ci-dessus, l'optimiseur a vu qu'il faut parcourir 577*1 lignes !!

Important : Pour les jointure externes, LEFT JOIN et RIGHT JOIN, l'ordre est important.

Indexes

USE INDEX

Il est possible d'indiquer à MariaDB d'utiliser un index spécifique ou d'ignorer un index :

```
MariaDB [(none)]> EXPLAIN SELECT email FROM customer INNER JOIN rental USE INDEX (idx_fk_customer_id) ON
customer.customer_id = rental.customer_id WHERE rental_date > '2006-04-10 00:00:00' AND rental_date < '2006-05-24
00:00:00'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: customer
         type: ALL
possible_keys: PRIMARY
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 577
        Extra:
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: rental
         type: ref
```

```
possible_keys: idx_fk_customer_id
          key: idx_fk_customer_id
      key_len: 2
          ref: sakila.customer.customer_id
         rows: 13
        Extra: Using where
2 rows in set (0.00 sec)
```

Important : Dans le cas d'utilisation de la clause USE INDEX, l'auteur de la requête indique à l'optimiseur d'utiliser l'index passé en paramètre. Cependant l'optimiseur peut décider d'utiliser un autre index si celui-si s'avère plus approprié. Dans le cas ci-dessus l'optimiseur a été d'accord avec l'auteur.

FORCE INDEX

```
MariaDB [(none)]> EXPLAIN SELECT email FROM customer INNER JOIN rental FORCE INDEX (idx_fk_customer_id) ON
customer.customer_id = rental.customer_id WHERE rental_date > '2006-04-10 00:00:00' AND rental_date < '2006-05-24
00:00:00'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: customer
         type: ALL
possible_keys: PRIMARY
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 577
        Extra:
*************************** 2. row ***************************
```

```
           id: 1
  select_type: SIMPLE
        table: rental
         type: ref
possible_keys: idx_fk_customer_id
          key: idx_fk_customer_id
      key_len: 2
          ref: sakila.customer.customer_id
         rows: 13
        Extra: Using where
2 rows in set (0.01 sec)
```

Important : Dans le cas d'utilisation de la clause FORCE INDEX, l'auteur de la requête indique à l'optimiseur d'utiliser l'index passé en paramètre. L'index sera utilisé sauf dans le cas où il n'est pas utilisable.

IGNORE INDEX

```
MariaDB [(none)]> EXPLAIN SELECT email FROM customer INNER JOIN rental IGNORE INDEX
(idx_fk_customer_id,rental_date) ON customer.customer_id = rental.customer_id WHERE rental_date > '2006-04-10
00:00:00' AND rental_date < '2006-05-24 00:00:00'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: rental
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
```

```
         rows: 15629
        Extra: Using where
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: customer
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 2
          ref: sakila.rental.customer_id
         rows: 1
        Extra:
2 rows in set (0.01 sec)

MariaDB [(none)]>
```

Important : Dans le cas d'utilisation de la clause IGNORE INDEX, l'auteur de la requête indique à l'optimiseur de ne **pas** utiliser le ou les index passés en paramètre. De ce fait notez l'inversion de l'ordre de jointure.

CLAUSES LENTES

Les clauses d'agrégation telles SUM(), MAX() et MIN() provoquent en générale des requêtes lentes car il faut lire l'ensemble de la table pour obtenir un résultat. C'est le même cas pour la clause COUNT() avec le moteur InnoDB, mais pas avec le moteur MyISAM. Le moteur MyISAM stocke dans ces métadonnées le nombre d'enregistrements de la table :

```
MariaDB [(none)]> SHOW TABLE STATUS WHERE Name = 'film_text'\G
*************************** 1. row ***************************
           Name: film_text
```

```
         Engine: MyISAM
        Version: 10
     Row_format: Dynamic
           Rows: 1000
 Avg_row_length: 119
    Data_length: 119616
Max_data_length: 281474976710655
   Index_length: 205824
      Data_free: 0
 Auto_increment: NULL
    Create_time: 2014-02-01 11:48:00
    Update_time: 2014-02-01 11:48:16
     Check_time: NULL
      Collation: utf8_general_ci
       Checksum: NULL
 Create_options:
        Comment:
1 row in set (0.00 sec)

MariaDB [(none)]> EXPLAIN SELECT COUNT(*) FROM film_text\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: NULL
         type: NULL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: NULL
        Extra: Select tables optimized away
1 row in set (0.00 sec)

MariaDB [(none)]>
```

Important : Dans le cas ci-dessus la valeur de la colonne **type** est **NULL** indiquant qu'aucun accès à la table n'est nécessaire.

Sous-requêtes

MariaDB exécute parfois très lentement des sous-requêtes. Privilégiez donc les jointures plutôt que des sous-requêtes :

```
MariaDB [(none)]> USE world;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [(none)]> SELECT Language FROM CountryLanguage WHERE CountryCode = (SELECT Code FROM Country WHERE
Name='France');
+------------+
| Language   |
+------------+
| Arabic     |
| French     |
| Italian    |
| Portuguese |
| Spanish    |
| Turkish    |
+------------+
6 rows in set (0.03 sec)

MariaDB [(none)]> SELECT Language FROM CountryLanguage INNER JOIN Country ON Code=CountryCode WHERE
Name='France';
+------------+
| Language   |
```

```
+------------+
| Arabic     |
| French     |
| Italian    |
| Portuguese |
| Spanish    |
| Turkish    |
+------------+
6 rows in set (0.01 sec)

MariaDB [(none)]>
```

Moteurs

MyISAM

Le paramètre le plus important pour le moteur MyISAM est le key_buffer_size :

```
MariaDB [(none)]> SHOW GLOBAL VARIABLES LIKE 'key_buffer_size';
+-----------------+---------+
| Variable_name   | Value   |
+-----------------+---------+
| key_buffer_size | 8388608 |
+-----------------+---------+
1 row in set (0.00 sec)

MariaDB [(none)]> SHOW GLOBAL STATUS LIKE 'Key_%';
+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| Key_blocks_not_flushed | 0     |
| Key_blocks_unused      | 7239  |
```

```
| Key_blocks_used         | 9     |
| Key_read_requests       | 39    |
| Key_reads               | 9     |
| Key_write_requests      | 0     |
| Key_writes              | 0     |
+-------------------------+-------+
7 rows in set (0.01 sec)

MariaDB [(none)]>
```

Important : La variable **Key_reads** indique le nombre de requêtes de lectures qui n'ont pas pu être satisfaites par le cache. La variable **Key_read_requests** indique le nombre de lectures totale. La formule (1 - Key_reads / Key_read_requests) * 100 donne une indication de l'efficacité du cache. Dans notre cas la valeur est de 76,92%. Une valeur basse indique qu'il faut augmenter la taille de key_buffer_size.

InnoDB

Les paramètres les plus importants pour le moteur **InnoDB** sont résumés dans la table suivante :

Paramètre	**Valeur Suggérée**
innodb_buffer_pool_size	80% de la mémoire physique de la machine
innodb_flush_log_at_trx_commit	1 si la protection des données est primordiale, 2 si la performance est optimale

Partitionnement

Le partitionnement est utilisé pour diviser les données d'une table en parties logiques selon une règle pré-établie :

- Par les données contenues dans la table. Dans ce cas on parle de partitionnement **horizontal**. Il existe 4 types possibles :

    - **Par plages**,
    - **Par listes**,
    - **Par hachage**,
    - **Par clef**,
- Par les colonnes de la table. Dans ce cas on parle de partitionnement **vertical**.

> **Important** : Il est possible de faire un partitionnement vertical et un partitionnement horizontal sur la même table. Il est aussi possible de faire des partitions de partitions.

Le partitionnement apporte trois avantages :

- Créer des tables plus grandes que la taille autorisée par un système de fichiers. Il est aussi possible de stocker des partitions à des endroits différents,
- Supprimer très rapidement des données en détruisant la partition qui les contient. Ceci s'appelle le **Scaling Back**,
- Optimiser certaines requêtes car les données étant organisées dans différentes partitions. Ceci s'appelle le **Partition Pruning**.

Partitionnement Horizontal

LAB #2 - Partitionnement par Plages

Créez une base de données **transactions** ainsi que la table **transac** :

```
CREATE DATABASE transactions;

USE transactions;

CREATE TABLE transac
(
id INT UNSIGNED PRIMARY KEY,
```

```
montant INT UNSIGNED NOT NULL,
jour DATE NOT NULL,
codePays ENUM('FR', 'BE', 'UK', 'US', 'CA', 'JP') NOT NULL
);
```

```
MariaDB [(none)]> CREATE DATABASE transactions;
Query OK, 1 row affected (0.00 sec)

MariaDB [(none)]> USE transactions;
Database changed
MariaDB [(none)]>

MariaDB [(none)]> CREATE TABLE transac
    -> (
    ->    id INT UNSIGNED PRIMARY KEY,
    ->    montant INT UNSIGNED NOT NULL,
    ->    jour DATE NOT NULL,
    ->    codePays ENUM('FR', 'BE', 'UK', 'US', 'CA', 'JP') NOT NULL
    -> );
Query OK, 0 rows affected (0.04 sec)

MariaDB [(none)]>
```

Imaginons qu'en étudiant les requêtes des utilisateurs sur cette table sur une période probante, nous découvrons que la majorité utilise des plages de date. Dans ce cas il serait utile de partitionner la table par la valeur de la colonne **jour**. Créer donc la table partitionnée appelée **transac_part** :

```
CREATE TABLE transac_part
(
id INT UNSIGNED NOT NULL,
montant INT UNSIGNED NOT NULL,
jour DATE NOT NULL,
codePays ENUM('FR', 'BE', 'UK', 'US', 'CA', 'JP') NOT NULL
) PARTITION BY RANGE(YEAR(jour))
(
```

```
PARTITION p1 VALUES LESS THAN(1997),
PARTITION p2 VALUES LESS THAN(1998),
PARTITION p3 VALUES LESS THAN(1999),
PARTITION p4 VALUES LESS THAN(2000),
PARTITION p5 VALUES LESS THAN(2001),
PARTITION p6 VALUES LESS THAN(2002),
PARTITION p7 VALUES LESS THAN(2003),
PARTITION p8 VALUES LESS THAN(2004),
PARTITION p9 VALUES LESS THAN(2005),
PARTITION p10 VALUES LESS THAN(2006),
PARTITION p11 VALUES LESS THAN MAXVALUE
);
```

```
MariaDB [(none)]> CREATE TABLE transac_part
    -> (
    ->    id INT UNSIGNED NOT NULL,
    ->    montant INT UNSIGNED NOT NULL,
    ->    jour DATE NOT NULL,
    ->    codePays ENUM('FR', 'BE', 'UK', 'US', 'CA', 'JP') NOT NULL
    -> ) PARTITION BY RANGE(YEAR(jour))
    -> (
    ->    PARTITION p1 VALUES LESS THAN(1997),
    ->    PARTITION p2 VALUES LESS THAN(1998),
    ->    PARTITION p3 VALUES LESS THAN(1999),
    ->    PARTITION p4 VALUES LESS THAN(2000),
    ->    PARTITION p5 VALUES LESS THAN(2001),
    ->    PARTITION p6 VALUES LESS THAN(2002),
    ->    PARTITION p7 VALUES LESS THAN(2003),
    ->    PARTITION p8 VALUES LESS THAN(2004),
    ->    PARTITION p9 VALUES LESS THAN(2005),
    ->    PARTITION p10 VALUES LESS THAN(2006),
    ->    PARTITION p11 VALUES LESS THAN MAXVALUE
    -> );
Query OK, 0 rows affected (0.13 sec)
```

```
MariaDB [(none)]>
```

L'ordre de définition des partitions est important. Les informations liées aux partitions sont stockées dans la table **information_schema.partitions**.

Créez ensuite une procédure pour injecter des données dans la table **transac** :

```
DELIMITER //
CREATE PROCEDURE remplir_transaction(nbTransacs INT)
BEGIN
DECLARE i INT DEFAULT 1;
DECLARE nbAlea DOUBLE;
DECLARE _jour DATE;
DECLARE _montant INT UNSIGNED;
DECLARE _codePays TINYINT UNSIGNED;

WHILE i <= nbTransacs DO
SET nbAlea = RAND();
SET _jour = ADDDATE('1996-01-01', FLOOR(nbAlea * 4015));
SET _montant = FLOOR(1 + (nbAlea * 9999));
SET nbAlea = RAND();
SET _codePays = FLOOR(1 + (nbAlea * 6));
INSERT INTO transac (id, montant, jour, codePays) VALUES (i, _montant, _jour, _codePays);
SET i = i + 1;
END WHILE;
END //

DELIMITER ;
```

```
MariaDB [(none)]> DELIMITER //
MariaDB [(none)]> CREATE PROCEDURE remplir_transaction(nbTransacs INT)
```

```
    -> BEGIN
    ->    DECLARE i INT DEFAULT 1;
    ->    DECLARE nbAlea DOUBLE;
    ->    DECLARE _jour DATE;
    ->    DECLARE _montant INT UNSIGNED;
    ->    DECLARE _codePays TINYINT UNSIGNED;
    ->
    ->    WHILE i <= nbTransacs DO
    ->       SET nbAlea = RAND();
    ->       SET _jour = ADDDATE('1996-01-01', FLOOR(nbAlea * 4015));
    ->       SET _montant = FLOOR(1 + (nbAlea * 9999));
    ->       SET nbAlea = RAND();
    ->       SET _codePays = FLOOR(1 + (nbAlea * 6));
    ->       INSERT INTO transac (id, montant, jour, codePays) VALUES (i, _montant, _jour, _codePays);
    ->       SET i = i + 1;
    ->    END WHILE;
    -> END //
Query OK, 0 rows affected (0.05 sec)

MariaDB [(none)]> DELIMITER ;
MariaDB [(none)]>
```

Appelez ensuite la procédure et patientez 10 minutes. A l'issue de la période, contrôlez le nombre d'enregistrements créés en sachant qu'il faut au mois 150 000 pour pouvoir voir l'impact de partitionnement :

```
MariaDB [(none)]> CALL remplir_transaction(200000);
^CCtrl-C -- sending "KILL QUERY 2" to server ...
Ctrl-C -- query aborted.
MariaDB [(none)]> SELECT COUNT(*) FROM transac;
+----------+
| COUNT(*) |
+----------+
|   191939 |
+----------+
```

```
1 row in set (0.09 sec)

MariaDB [(none)]>

Insérez maintenant les mêmes données dans la table **transac_part** :

<code>
MariaDB [(none)]> INSERT INTO transac_part SELECT * FROM transac;
Query OK, 191939 rows affected (3.63 sec)
Records: 191939  Duplicates: 0  Warnings: 0

MariaDB [(none)]> SELECT COUNT(*) FROM transac_part;
+----------+
| COUNT(*) |
+----------+
|   191939 |
+----------+
1 row in set (0.14 sec)

MariaDB [(none)]>
```

Pour voir l'impact du partitionnement sur la performance, saisissez les requêtes suivantes :

```
MariaDB [(none)]> SELECT SUM(montant), AVG(montant) FROM transac WHERE jour BETWEEN '1998-01-01' AND
'1998-12-31';
+--------------+--------------+
| SUM(montant) | AVG(montant) |
+--------------+--------------+
|     39970960 |    2276.8989 |
+--------------+--------------+
1 row in set (0.17 sec)

MariaDB [(none)]> SELECT SUM(montant), AVG(montant) FROM transac_part WHERE jour BETWEEN '1998-01-01' AND
'1998-12-31';
```

```
+--------------+--------------+
| SUM(montant) | AVG(montant) |
+--------------+--------------+
|     39970960 |    2276.8989 |
+--------------+--------------+
1 row in set (0.03 sec)
```

Notez que la requête sur la table partitionnée retourne un résultat **5,67** fois plus vite que sur la table d'origine. Ceci est du au fait que le serveur exclue les partitions qui ne contiennent pas les données recherchées. Ceci peut être constaté en utilisant la clause EXPLAIN :

```
MariaDB [(none)]> EXPLAIN PARTITIONS SELECT SUM(montant), AVG(montant) FROM transac_part WHERE jour BETWEEN
'1998-01-01' AND '1998-12-31'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: transac_part
   partitions: p3
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 18035
        Extra: Using where
1 row in set (0.01 sec)

MariaDB [(none)]>
```

Important : Notez que la valeur de la colonne **partitions** est **p3**.

Evidemment, plus qu'il y a de partitions à consulter, moins important est le gains en performance :

```
MariaDB [(none)]> SELECT SUM(montant), AVG(montant) FROM transac WHERE jour BETWEEN '1996-01-01' AND
'2000-12-31';
+--------------+--------------+
| SUM(montant) | AVG(montant) |
+--------------+--------------+
|    198031481 |    2268.9216 |
+--------------+--------------+
1 row in set (0.18 sec)

MariaDB [(none)]> SELECT SUM(montant), AVG(montant) FROM transac_part WHERE jour BETWEEN '1996-01-01' AND
'2000-12-31';
+--------------+--------------+
| SUM(montant) | AVG(montant) |
+--------------+--------------+
|    198031481 |    2268.9216 |
+--------------+--------------+
1 row in set (0.16 sec)

MariaDB [(none)]>
```

Le système de partitionnement peut aussi s'avérer un handicap au niveau de la performance si les requêtes portent sur l'ensemble des données :

```
MariaDB [(none)]> SELECT SUM(montant), AVG(montant) FROM transac;
+--------------+--------------+
| SUM(montant) | AVG(montant) |
+--------------+--------------+
|    959746672 |    5000.2692 |
+--------------+--------------+
1 row in set (0.16 sec)

MariaDB [(none)]> SELECT SUM(montant), AVG(montant) FROM transac_part;
+--------------+--------------+
| SUM(montant) | AVG(montant) |
+--------------+--------------+
```

```
|    959746672 |    5000.2692 |
+--------------+--------------+
1 row in set (0.20 sec)

MariaDB [(none)]>
```

Dernièrement, l'effacement de données en grande quantité est plus rapide quand on drop une partition :

```
MariaDB [(none)]> DELETE FROM transac_part WHERE jour BETWEEN '1996-01-01' AND '1996-12-31';
Query OK, 17680 rows affected (0.18 sec)

MariaDB [(none)]> ALTER TABLE transac_part DROP PARTITION p1;
Query OK, 0 rows affected (0.10 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [(none)]>
```

LAB #3 - Partitionnement par Listes

Imaginons maintenant que nous avons découvert que les requêtes des utilisateurs portent souvent sur continent géographique. Dans ce cas, il serait utile de faire un partitionnement par liste. Le champs crée par l'instruction suivante :

```
codePays ENUM('FR', 'BE', 'UK', 'US', 'CA', 'JP') NOT NULL
```

associe une valeur entière à chaque valeur du champ de type ENUM. Nous allons utiliser ce fait pour créer la table partitionnée.

Commencez par nettoyer ce que vous avez déjà fait :

```
MariaDB [(none)]> TRUNCATE TABLE transac;
Query OK, 0 rows affected (0.17 sec)

MariaDB [(none)]> DROP TABLE transac_part;
```

```
Query OK, 0 rows affected (0.09 sec)

MariaDB [(none)]>
```

Créez maintenant la table transac_part :

```
MariaDB [(none)]> CREATE TABLE transac_part
    -> (
    ->    id INT UNSIGNED NOT NULL,
    ->    montant INT UNSIGNED NOT NULL,
    ->    jour DATE NOT NULL,
    ->    codePays TINYINT UNSIGNED NOT NULL
    -> ) PARTITION BY LIST(codePays)
    -> (
    ->    PARTITION pEurope VALUES IN (1, 2, 3),
    ->    PARTITION pAmeriqueNord VALUES IN (4, 5),
    ->    PARTITION pAsie VALUES IN (6)
    -> );
Query OK, 0 rows affected (0.06 sec)
```

Appelez la procédure remplir_transaction et copiez la valeurs dans transac_part :

```
MariaDB [(none)]> CALL remplir_transaction(10000000);
^CCtrl-C -- sending "KILL QUERY 2" to server ...
Ctrl-C -- query aborted.
MariaDB [(none)]> INSERT INTO transac_part SELECT * FROM transac;
Query OK, 52193 rows affected (0.57 sec)
Records: 52193  Duplicates: 0  Warnings: 0

MariaDB [(none)]>
```

Comparez le gains de performance en utilisant la table partitionnée :

```
MariaDB [(none)]> SELECT SUM(montant), AVG(montant) FROM transac WHERE codePays IN ('FR','BE','UK');
```

```
+--------------+--------------+
| SUM(montant) | AVG(montant) |
+--------------+--------------+
|    130838834 |    5018.9433 |
+--------------+--------------+
1 row in set (0.07 sec)

MariaDB [(none)]> SELECT SUM(montant), AVG(montant) FROM transac_part WHERE codePays IN (1, 2, 3);
+--------------+--------------+
| SUM(montant) | AVG(montant) |
+--------------+--------------+
|    130838834 |    5018.9433 |
+--------------+--------------+
1 row in set (0.04 sec)

MariaDB [(none)]>
```

Notez que la requête sur la table partitionnée retourne un résultat **1,75** fois plus vite que sur la table d'origine.

Comparez le gains de performance en utilisant uniquement la partition numéro 6 :

```
MariaDB [(none)]> SELECT SUM(montant), AVG(montant) FROM transac WHERE codePays = 'JP';
+--------------+--------------+
| SUM(montant) | AVG(montant) |
+--------------+--------------+
|     43098698 |    4973.3093 |
+--------------+--------------+
1 row in set (0.05 sec)

MariaDB [(none)]> SELECT SUM(montant), AVG(montant) FROM transac_part WHERE codePays = 6;
+--------------+--------------+
| SUM(montant) | AVG(montant) |
+--------------+--------------+
|     43098698 |    4973.3093 |
```

```
+--------------+--------------+
1 row in set (0.01 sec)

MariaDB [(none)]>
```

Notez que la requête sur la table partitionnée retourne un résultat **5** fois plus vite que sur la table d'origine. Ceci est du au fait qu'il y a moins d'enregistrements à parcourir.

LAB #4 - Partitionnement par Hash

Dans ce cas la partition à laquelle appartient un enregistrement est déterminée à partir de la valeur de retour d'une fonction définie par l'utilisateur.

Re-créez la table transac_part :

```
MariaDB [(none)]> DROP TABLE transac_part;
Query OK, 0 rows affected (0.03 sec)

MariaDB [(none)]>
MariaDB [(none)]> CREATE TABLE transac_part
    -> (
    ->    id INT UNSIGNED NOT NULL,
    ->    montant INT UNSIGNED NOT NULL,
    ->    jour DATE NOT NULL,
    ->    codePays ENUM('FR', 'BE', 'UK', 'US', 'CA', 'JP') NOT NULL
    -> ) PARTITION BY HASH(YEAR(jour)) PARTITIONS 11;
Query OK, 0 rows affected (0.07 sec)

MariaDB [(none)]>
MariaDB [(none)]>
MariaDB [(none)]> INSERT INTO transac_part SELECT * FROM transac;
Query OK, 52193 rows affected (0.85 sec)
Records: 52193  Duplicates: 0  Warnings: 0
```

```
MariaDB [(none)]>
```

Important : Notez qu'ici on indique le champs de référence **jour** et le nombre de partitions désirées.

Testez maintenant le gains de performance en utilisant les partitions :

```
MariaDB [(none)]> SELECT SUM(montant), AVG(montant) FROM transac WHERE jour BETWEEN '1998-01-01' AND
'1998-12-31';
+--------------+--------------+
| SUM(montant) | AVG(montant) |
+--------------+--------------+
|     10867590 |    2271.6534 |
+--------------+--------------+
1 row in set (0.09 sec)

MariaDB [(none)]> SELECT SUM(montant), AVG(montant) FROM transac_part WHERE jour BETWEEN '1998-01-01' AND
'1998-12-31';
+--------------+--------------+
| SUM(montant) | AVG(montant) |
+--------------+--------------+
|     10867590 |    2271.6534 |
+--------------+--------------+
1 row in set (0.08 sec)

MariaDB [(none)]>
```

Important : Notez qu'ici le gains de performance est minime.

Le gains de performance est minime parce que toutes les partitions ont été scannées :

```
MariaDB [(none)]> EXPLAIN PARTITIONS SELECT SUM(montant), AVG(montant) FROM transac_part WHERE jour BETWEEN
'1998-01-01' AND '1998-12-31'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: transac_part
   partitions: p0,p1,p2,p3,p4,p5,p6,p7,p8,p9,p10
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 52982
        Extra: Using where
1 row in set (0.00 sec)

MariaDB [(none)]>
```

Important : Le partitionnement par hachage n'est efficace que dans le cas ou le hash est généré à partir d'une colonne de type entier.

Pour prouver ce point, recréez la table transac_part avec des partitions basées sur la colonne **codePays** :

```
MariaDB [(none)]> DROP TABLE transac_part;
Query OK, 0 rows affected (0.22 sec)

MariaDB [(none)]>
MariaDB [(none)]> CREATE TABLE transac_part
    -> (
    ->    id INT UNSIGNED NOT NULL,
    ->    montant INT UNSIGNED NOT NULL,
```

```
    ->    jour DATE NOT NULL,
    ->    codePays TINYINT UNSIGNED NOT NULL
    -> ) PARTITION BY HASH(codePays) PARTITIONS 6;
Query OK, 0 rows affected (0.23 sec)

MariaDB [(none)]>
MariaDB [(none)]> INSERT INTO transac_part SELECT * FROM transac;
Query OK, 52193 rows affected (0.65 sec)
Records: 52193  Duplicates: 0  Warnings: 0

MariaDB [(none)]>
```

Testez de nouveau le gain de performance en utilisant les partitions :

```
MariaDB [(none)]> SELECT SUM(montant), AVG(montant) FROM transac WHERE codePays = 1;
+--------------+--------------+
| SUM(montant) | AVG(montant) |
+--------------+--------------+
|     43320506 |    5023.2498 |
+--------------+--------------+
1 row in set (0.04 sec)

MariaDB [(none)]> SELECT SUM(montant), AVG(montant) FROM transac_part WHERE codePays = 1;
+--------------+--------------+
| SUM(montant) | AVG(montant) |
+--------------+--------------+
|     43320506 |    5023.2498 |
+--------------+--------------+
1 row in set (0.02 sec)

MariaDB [(none)]>
```

Important : Cette fois-ci le gain est plus important.

Partitionnement par Key

Dans ce cas la partition à laquelle appartient un enregistrement est déterminée à partir de la valeur de retour d'une fonction définie par le serveur MariaDB.

Par exemple :

```
CREATE TABLE transac_part
(
   id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
   montant INT UNSIGNED NOT NULL,
   jour DATE NOT NULL,
   codePays ENUM('FR', 'BE', 'UK', 'US', 'CA', 'JP') NOT NULL
) PARTITION BY KEY() PARTITIONS 5;
```

Important : Dans ce cas, la clef n'est pas spécifiée. Le hachage est généré à partir de tous les champs de la clef primaire. Dans notre cas c'est le champ **id**. Puisqu'il y a 5 partitions, si le nombre de données est de 100 000, le serveur mettra exactement 20 000 enregistrements dans chaque partition.

LAB #5 - Sous-partitionnement

Avec MariaDB il est possible de créer des sous-partitions. Cependant il existe deux limitations :

- La partition de premier niveau doit être de type RANGE ou LIST,

- La partition de deuxième niveau doit être de type HASH ou KEY,
- Le nombre de partitions total ne peut pas dépasser **8192**.

Prenons le cas où les requêtes fréquentes portent sur les dates **et** sur la zone géographique.

Créez la table transac-part :

```
MariaDB [(none)]> DROP TABLE transac_part;
Query OK, 0 rows affected (0.18 sec)

MariaDB [(none)]>
MariaDB [(none)]> CREATE TABLE transac_part
    -> (
    ->    id INT UNSIGNED NOT NULL,
    ->    montant INT UNSIGNED NOT NULL,
    ->    jour DATE NOT NULL,
    ->    codePays INT UNSIGNED NOT NULL
    -> )
    -> PARTITION BY RANGE(YEAR(jour))
    -> SUBPARTITION BY HASH(codePays) SUBPARTITIONS 6
    -> (
    ->    PARTITION p1 VALUES LESS THAN(1997),
    ->    PARTITION p2 VALUES LESS THAN(1998),
    ->    PARTITION p3 VALUES LESS THAN(1999),
    ->    PARTITION p4 VALUES LESS THAN(2000),
    ->    PARTITION p5 VALUES LESS THAN(2001),
    ->    PARTITION p6 VALUES LESS THAN(2002),
    ->    PARTITION p7 VALUES LESS THAN(2003),
    ->    PARTITION p8 VALUES LESS THAN(2004),
    ->    PARTITION p9 VALUES LESS THAN(2005),
    ->    PARTITION p10 VALUES LESS THAN(2006),
    ->    PARTITION p11 VALUES LESS THAN MAXVALUE
    -> );
Query OK, 0 rows affected (1.37 sec)
```

```
MariaDB [(none)]>
MariaDB [(none)]> INSERT INTO transac_part SELECT * FROM transac;
```

Testez maintenant le gain de performance en utilisant les partitions :

```
MariaDB [(none)]> SELECT SUM(montant), AVG(montant) FROM transac WHERE codePays=2 AND jour BETWEEN '2000-01-01'
AND '2000-12-01';
+--------------+--------------+
| SUM(montant) | AVG(montant) |
+--------------+--------------+
|      2928528 |    4073.0570 |
+--------------+--------------+
1 row in set (0.05 sec)

MariaDB [(none)]> SELECT SUM(montant), AVG(montant) FROM transac_part WHERE codePays=2 AND jour BETWEEN
'2000-01-01' AND '2000-12-01';
+--------------+--------------+
| SUM(montant) | AVG(montant) |
+--------------+--------------+
|      2928528 |    4073.0570 |
+--------------+--------------+
1 row in set (0.01 sec)

MariaDB [(none)]> SELECT SUM(montant), AVG(montant) FROM transac;
+--------------+--------------+
| SUM(montant) | AVG(montant) |
+--------------+--------------+
|    261035779 |    5001.3561 |
+--------------+--------------+
1 row in set (0.07 sec)

MariaDB [(none)]> SELECT SUM(montant), AVG(montant) FROM transac_part;
+--------------+--------------+
| SUM(montant) | AVG(montant) |
```

```
+--------------+--------------+
|    261035779 |    5001.3561 |
+--------------+--------------+
1 row in set (0.09 sec)

MariaDB [(none)]>
```

Notez la dégradation des performance avec les partitions dans le cas d'une requête sans clause WHERE :

```
MariaDB [(none)]> EXPLAIN PARTITIONS SELECT SUM(montant), AVG(montant) FROM transac_part WHERE codePays=2 AND
jour BETWEEN '2000-01-01' AND '2000-12-01'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: transac_part
   partitions: p5_p5sp2
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 893
        Extra: Using where
1 row in set (0.01 sec)

MariaDB [(none)]> EXPLAIN PARTITIONS SELECT SUM(montant), AVG(montant) FROM transac_part\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: transac_part
   partitions:
p1_p1sp0,p1_p1sp1,p1_p1sp2,p1_p1sp3,p1_p1sp4,p1_p1sp5,p2_p2sp0,p2_p2sp1,p2_p2sp2,p2_p2sp3,p2_p2sp4,p2_p2sp5,p3_p3
sp0,p3_p3sp1,p3_p3sp2,p3_p3sp3,p3_p3sp4,p3_p3sp5,p4_p4sp0,p4_p4sp1,p4_p4sp2,p4_p4sp3,p4_p4sp4,p4_p4sp5,p5_p5sp0,p
5_p5sp1,p5_p5sp2,p5_p5sp3,p5_p5sp4,p5_p5sp5,p6_p6sp0,p6_p6sp1,p6_p6sp2,p6_p6sp3,p6_p6sp4,p6_p6sp5,p7_p7sp0,p7_p7s
```

```
p1,p7_p7sp2,p7_p7sp3,p7_p7sp4,p7_p7sp5,p8_p8sp0,p8_p8sp1,p8_p8sp2,p8_p8sp3,p8_p8sp4,p8_p8sp5,p9_p9sp0,p9_p9sp1,p9
_p9sp2,p9_p9sp3,p9_p9sp4,p9_p9sp5,p10_p10sp0,p10_p10sp1,p10_p10sp2,p10_p10sp3,p10_p10sp4,p10_p10sp5,p11_p11sp0,p1
1_p11sp1,p11_p11sp2,p11_p11sp3,p11_p11sp4,p11_p11sp5
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 54365
        Extra:
1 row in set (0.01 sec)

MariaDB [(none)]>
```

LAB #6 - Partitionnement Vertical

Dans ce cas les partitions sont des tables différentes. Prenons l'exemple d'une base de données contenant des photos :

```
MariaDB [(none)]> CREATE TABLE produit
    -> (
    ->    id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    ->    libelle VARCHAR(50) NOT NULL,
    ->    prix DECIMAL(10,2) NOT NULL,
    ->    photo BLOB NOT NULL
    -> );
Query OK, 0 rows affected (0.04 sec)
```

Séparez maintenant les photos des autres données :

```
MariaDB [(none)]> CREATE TABLE produit2
    -> (
    ->    id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    ->    libelle VARCHAR(50) NOT NULL,
```

```
    ->    prix DECIMAL(10,2) NOT NULL
    -> );
Query OK, 0 rows affected (0.04 sec)

MariaDB [(none)]>
MariaDB [(none)]> CREATE TABLE photo_produit
    -> (
    ->    idProd INT UNIQUE NOT NULL,
    ->    photo BLOB NOT NULL
    -> );
Query OK, 0 rows affected (0.03 sec)

MariaDB [(none)]>
```

Utilisez la procédure suivante pour injecter les données :

```
MariaDB [(none)]> DELIMITER //
MariaDB [(none)]>
MariaDB [(none)]> CREATE PROCEDURE remplir_produit(nbProds INT)
    -> BEGIN
    ->    DECLARE i INT DEFAULT 1;
    ->    DECLARE _prix DECIMAL(10,2);
    ->    DECLARE _photo BLOB DEFAULT REPEAT('A', 20480);
    ->
    ->    WHILE i <= nbProds DO
    ->       SET _prix = CAST((RAND() * 1000) AS DECIMAL(10,2));
    ->       INSERT INTO produit VALUES (i, CONCAT('Produit_', i), _prix, _photo);
    ->       INSERT INTO produit2 VALUES (i, CONCAT('Produit_', i), _prix);
    ->       INSERT INTO photo_produit VALUES (i, _photo);
    ->       SET i = i + 1;
    ->    END WHILE;
    -> END //
Query OK, 0 rows affected (0.10 sec)
```

```
MariaDB [(none)]>
MariaDB [(none)]> DELIMITER ;
MariaDB [(none)]> CALL remplir_produit(10000);
^CCtrl-C -- sending "KILL QUERY 3" to server ...
Ctrl-C -- query aborted.
ERROR 1317 (70100): Query execution was interrupted
MariaDB [(none)]> SELECT COUNT(*) FROM produit;
+----------+
| COUNT(*) |
+----------+
|     1664 |
+----------+
1 row in set (0.02 sec)

MariaDB [(none)]> SELECT COUNT(*) FROM produit2;
+----------+
| COUNT(*) |
+----------+
|     1664 |
+----------+
1 row in set (0.01 sec)

MariaDB [(none)]> SELECT COUNT(*) FROM photo_produit;
+----------+
| COUNT(*) |
+----------+
|     1663 |
+----------+
1 row in set (0.01 sec)

MariaDB [(none)]>
```

Testez maintenant le gain de performance en utilisant les partitions :

```
MariaDB [(none)]> SELECT id, libelle, prix FROM produit;
MariaDB [(none)]> SELECT id, libelle, prix FROM produit2;
```
