ORSYS formation

Formation MariaDB

Animé par Michel BOCCIOLESI

Table des Matières

LES DATAS DÉFINITIONS

Nous allons voir les définitions de conception et de modélisation du langage SQL.

Le langage SQL (Structured Query Language) regroupe différentes parties de conception, de manipulation de données et de gestion des droits des utilisateurs .

1ère Partie : Le Data Definition Language : DDL

C'est la partie du langage qui permet de créer des bases de données, les tables, les index Elle regroupe les instructions **CREATE**, **ALTER**, **DROP** (créer, modifier, supprimer un élément de la base).

2ne Partie : Le Data Manipulation Language : DML

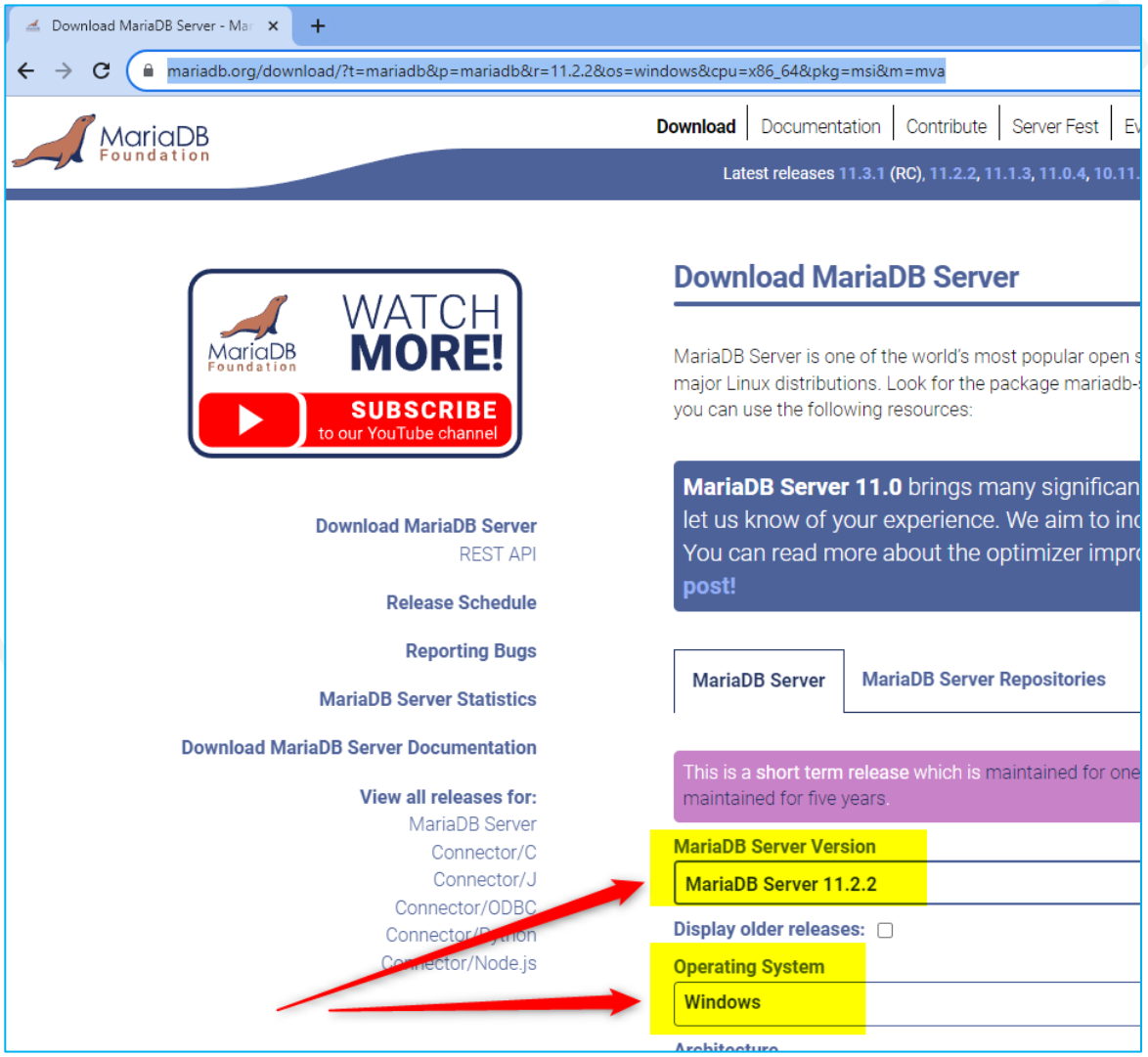
C'est la partie du langage qui traite les données.Elle regroupe les instructions **INSERT**, **UPDATE**, **DELETE**, **SELECT** (insertion, mise à jour, suppression et extraction de données).

3ème Partie : Le Data Control Language : DCL

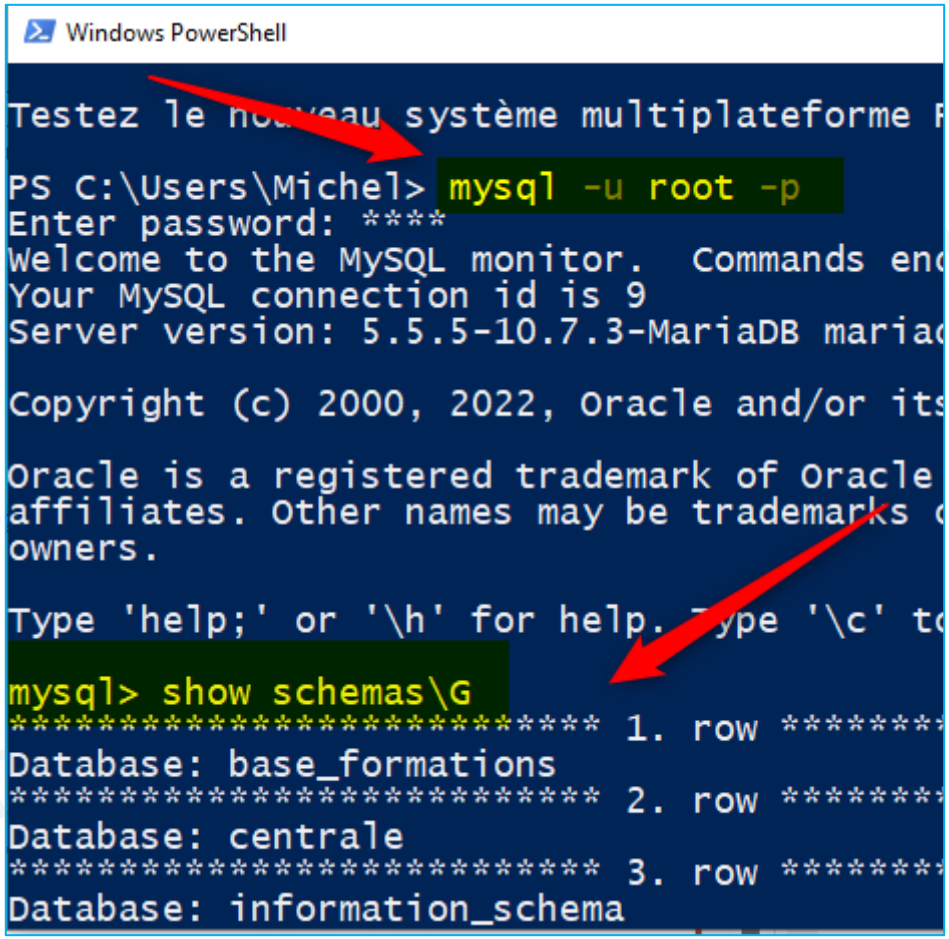
C'est la partie du SQL qui gère les droits d'accès aux tables. Elle regroupe les instructions **GRANT**, **REVOKE** (attribution et suppression des droits).

INSTALLATION

https://mariadb.org/download

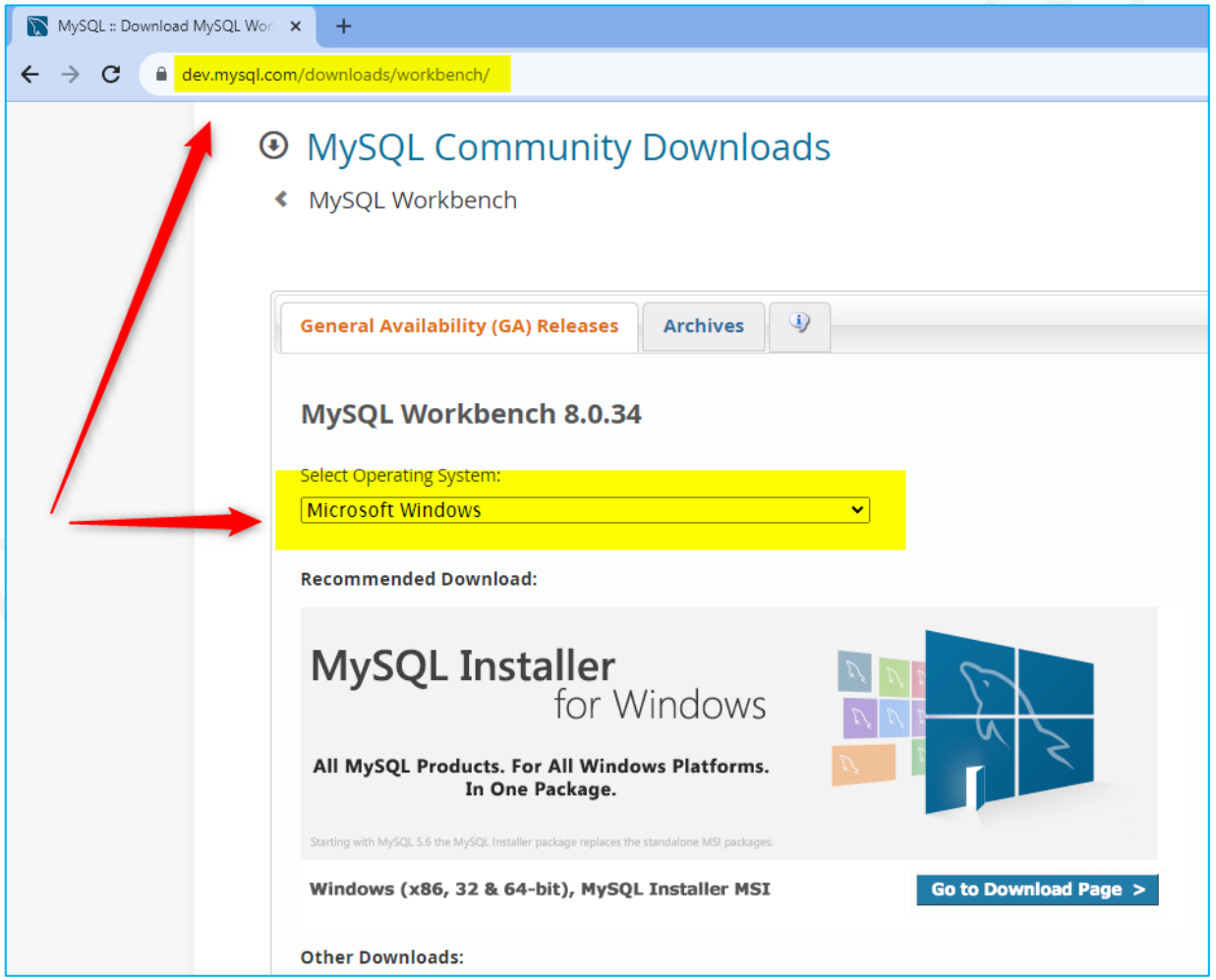


INSTALLATION

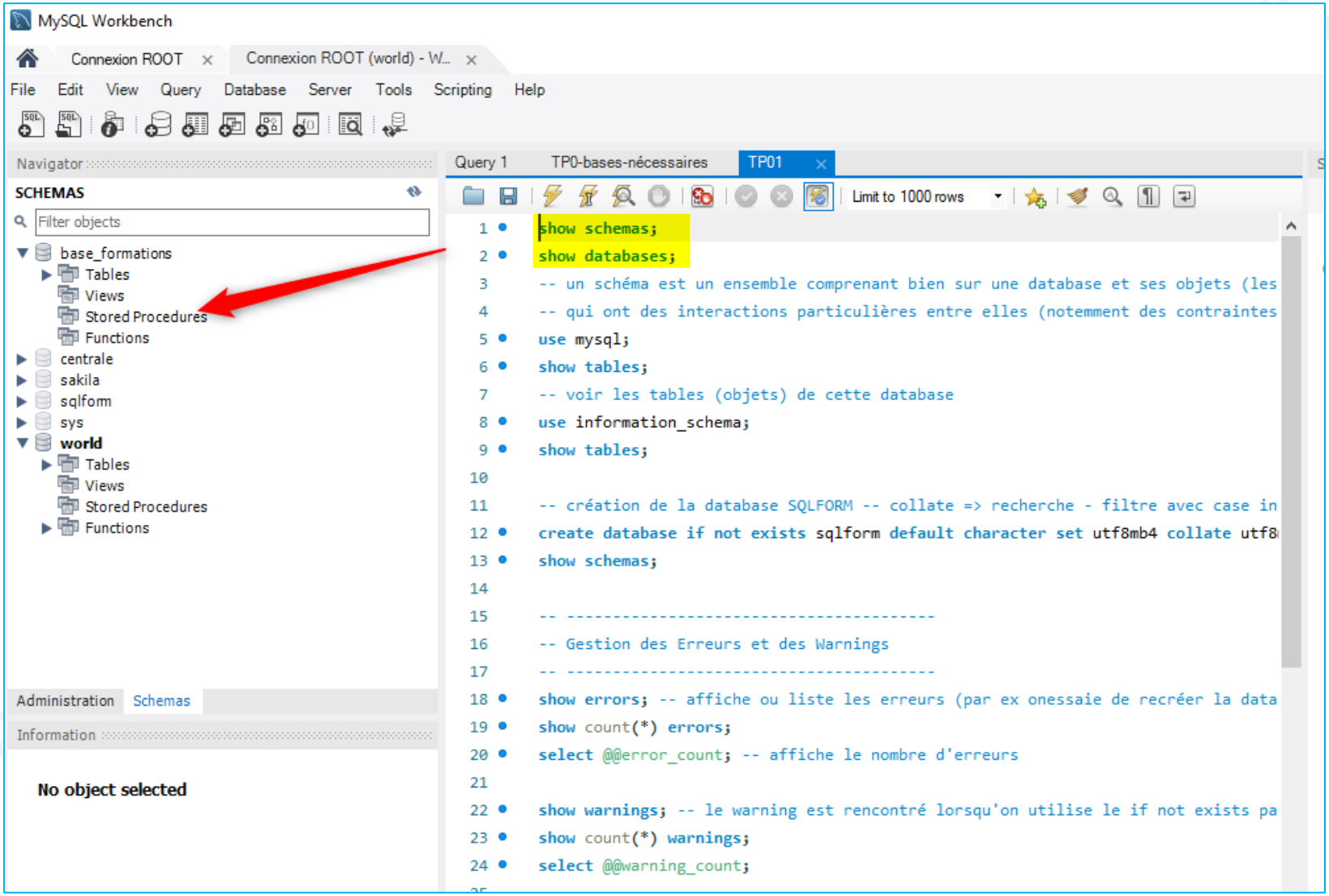


INSTALLATION

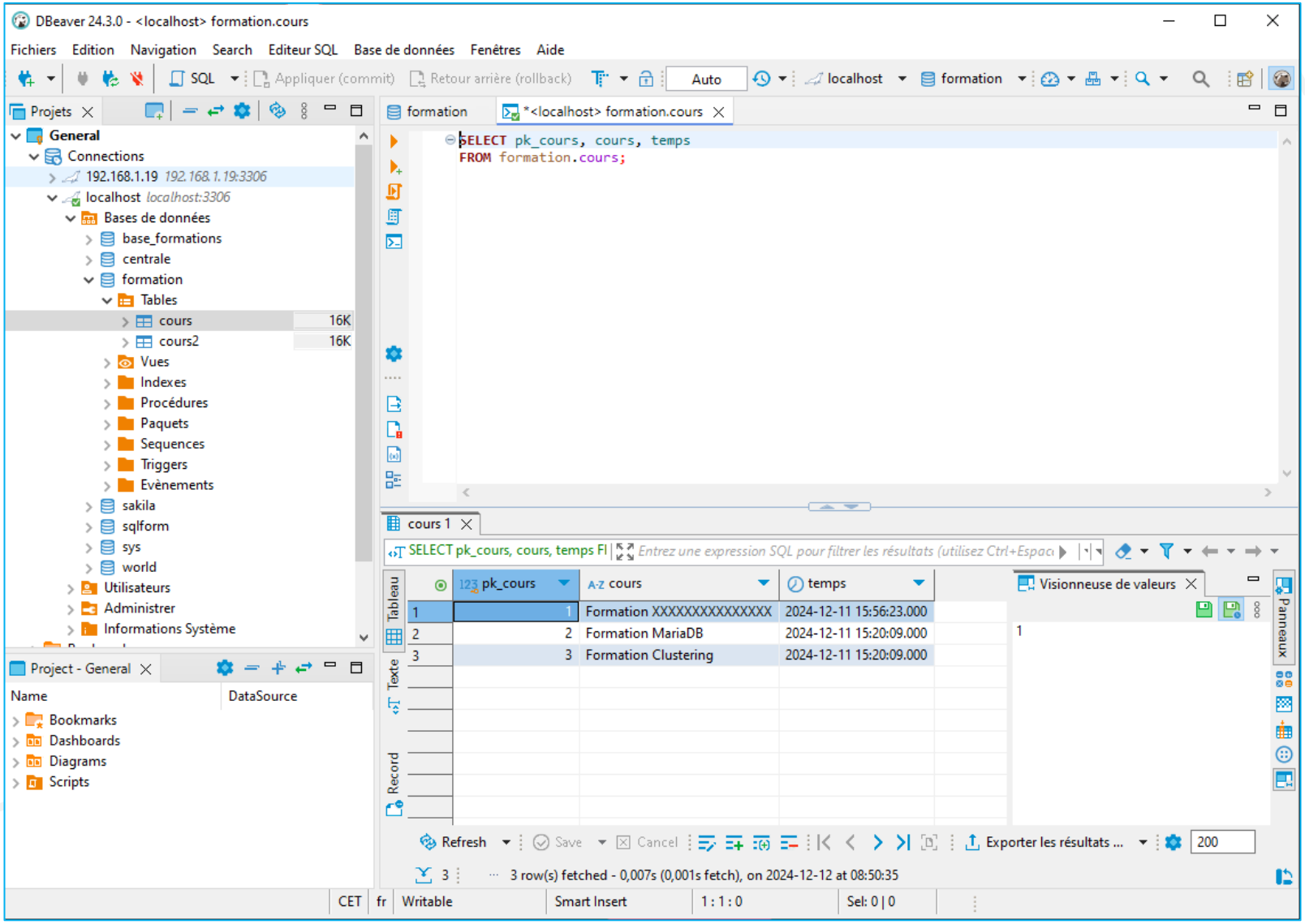
https://dev.mysql.com/downloads/workbench/



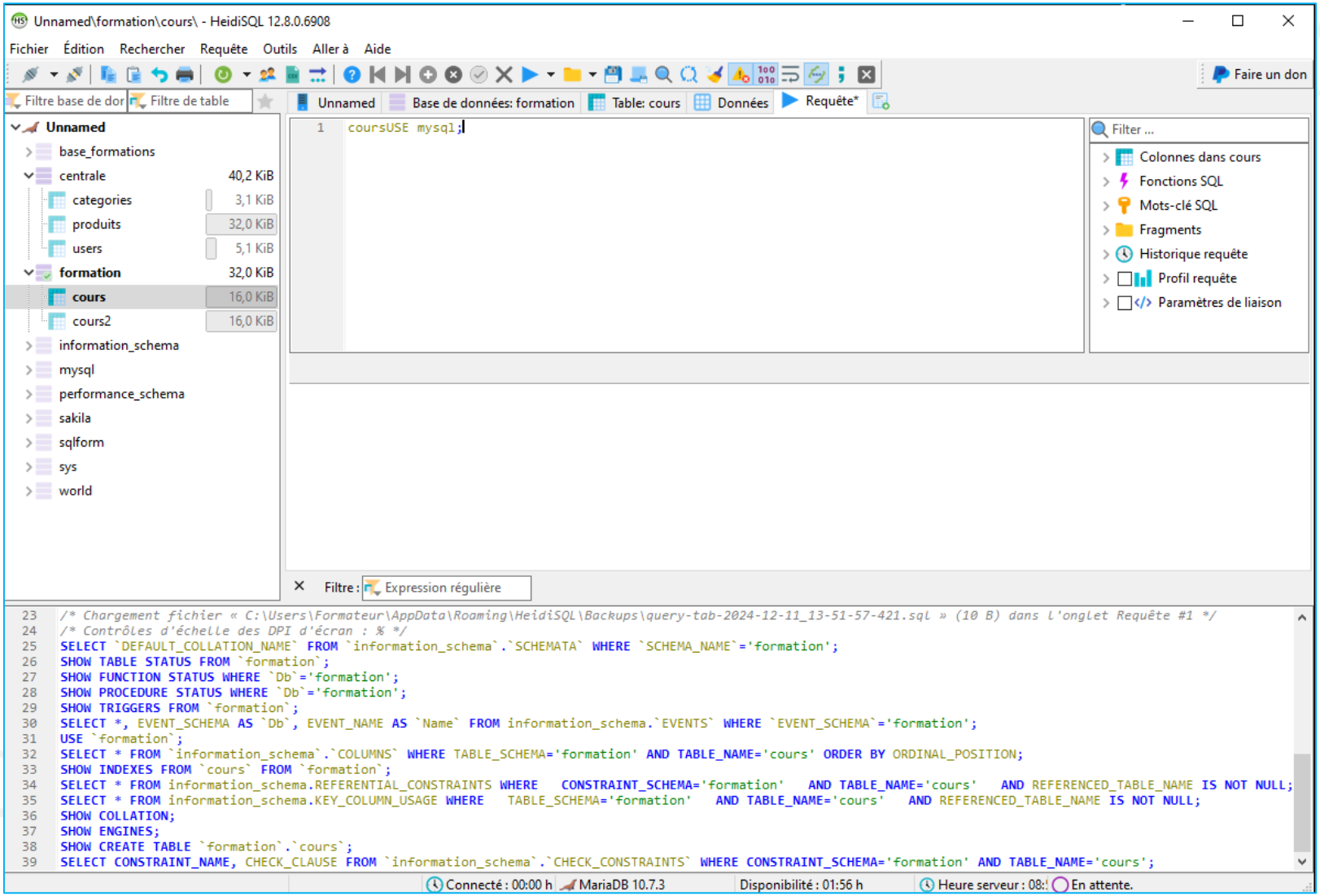
WORKBENCH


MySQL Workbench
Connexion ROOT
Connexion ROOT (world) - W...
File Edit View Query Database Server Tools Scripting Help
Navigator
SCHEMAS
Filter objects
base_formations
Tables
Views
Stored Procedures
Functions
centrale
sakila
sqlform
sys
world
Tables
Views
Stored Procedures
Functions
Administration Schemas
Information
No object selected
Query 1 TP0-bases-nécessaires TP01
Limit to 1000 rows
1 show schemas;
2 show databases;
3 -- un schéma est un ensemble comprenant bien sur une database et ses objets (les
4 -- qui ont des interactions particulières entre elles (notemment des contraintes
5 use mysql;
6 show tables;
7 -- voir les tables (objets) de cette database
8 use information_schema;
9 show tables;
10
11 -- création de la database SQLFORM -- collate => recherche - filtre avec case in
12 create database if not exists sqlform default character set utf8mb4 collate utf8
13 show schemas;
14
15 -- --
16 -- Gestion des Erreurs et des Warnings
17 -- --
18 show errors; -- affiche ou liste les erreurs (par ex onessaie de recréer la data
19 show count(*) errors;
20 select @@error_count; -- affiche le nombre d'erreurs
21
22 show warnings; -- le warning est rencontré lorsqu'on utilise le if not exists pa
23 show count(*) warnings;
24 select @@warning_count;

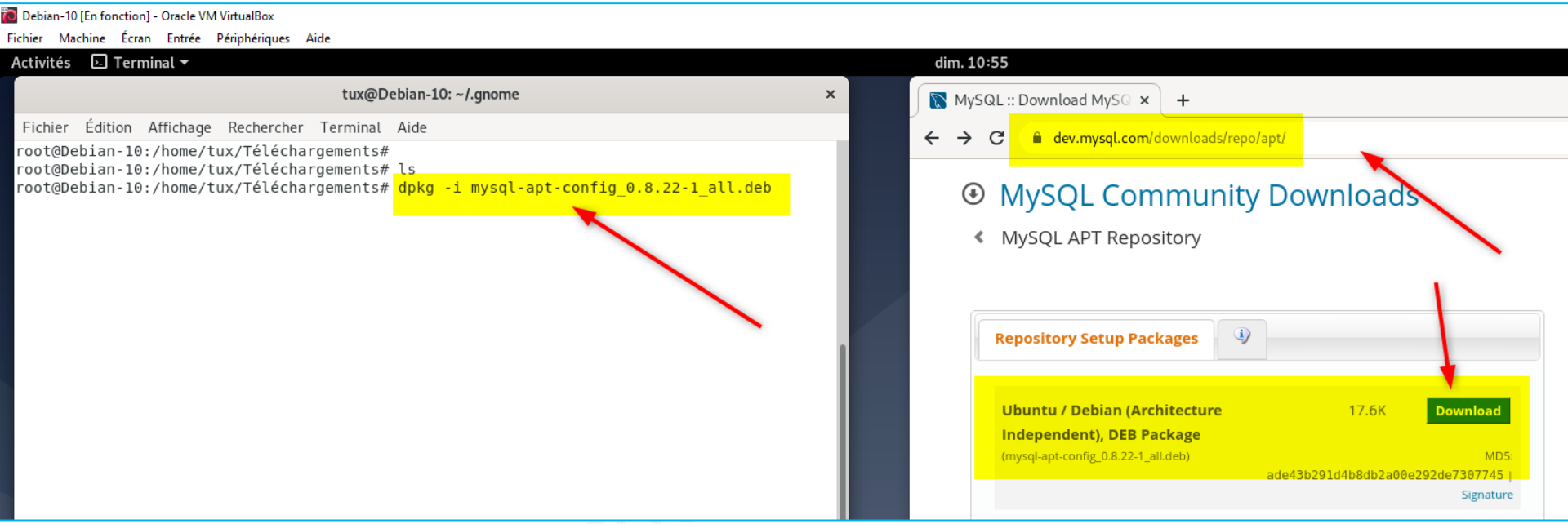
DBEAVER

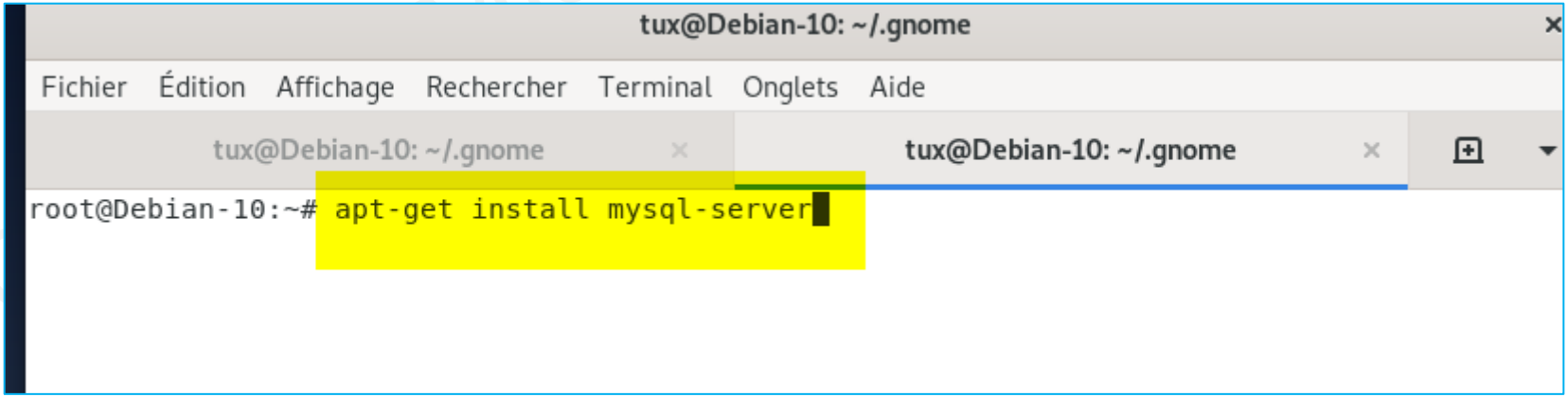

DBeaver 24.3.0 - <localhost> formation.cours
Fichiers Edition Navigation Search Editeur SQL Base de données Fenêtres Aide
SQL Appliquer (commit) Retour arrière (rollback) Auto localhost formation
Projets
General
Connections
192.168.1.19 192.168.1.19:3306
localhost localhost:3306
Bases de données
base_formations
centrale
formation
Tables
cours 16K
cours2 16K
Vues
Indexes
Procédures
Paquets
Sequences
Triggers
Evènements
sakila
sqlform
sys
world
Utilisateurs
Administrer
Informations Système
Project - General
Name DataSource
Bookmarks
Dashboards
Diagrams
Scripts
formation
*<localhost> formation.cours
SELECT pk_cours, cours, temps
FROM formation.cours;
cours 1
SELECT pk_cours, cours, temps Fl
Entrez une expression SQL pour filtrer les résultats (utilisez Ctrl+Espac
pk_cours
cours
temps
1 1 Formation XXXXXXXXXXXXXX 2024-12-11 15:56:23.000
2 2 Formation MariaDB 2024-12-11 15:20:09.000
3 3 Formation Clustering 2024-12-11 15:20:09.000
Visionneuse de valeurs
1
Tableau
Texte
Record
Panneaux
Refresh Save Cancel Exporter les résultats ... 200
3 row(s) fetched - 0,007s (0,001s fetch), on 2024-12-12 at 08:50:35
CET fr Writable Smart Insert 1 : 1 : 0 Sel: 0 | 0

HEIDISQL


Unnamed\formation\cours\ - HeidiSQL 12.8.0.6908
Fichier Édition Rechercher Requête Outils Aller à Aide
Faire un don
Filtre base de dor
Filtre de table
Unnamed
Base de données: formation
Table: cours
Données
Requête*
Unnamed
base_formations
centrale 40,2 KiB
categories 3,1 KiB
produits 32,0 KiB
users 5,1 KiB
formation 32,0 KiB
cours 16,0 KiB
cours2 16,0 KiB
information_schema
mysql
performance_schema
sakila
sqlform
sys
world
1 coursUSE mysql;
Filter ...
Colonnes dans cours
Fonctions SQL
Mots-clé SQL
Fragments
Historique requête
Profil requête
Paramètres de liaison
Filtre : Expression régulière
23 /* Chargement fichier « C:\Users\Formateur\AppData\Roaming\HeidiSQL\Backups\query-tab-2024-12-11_13-51-57-421.sql » (10 B) dans l'onglet Requête #1 */
24 /* Contrôles d'échelle des DPI d'écran : % */
25 SELECT `DEFAULT_COLLATION_NAME` FROM `information_schema`.`SCHEMATA` WHERE `SCHEMA_NAME`='formation';
26 SHOW TABLE STATUS FROM `formation`;
27 SHOW FUNCTION STATUS WHERE `Db`='formation';
28 SHOW PROCEDURE STATUS WHERE `Db`='formation';
29 SHOW TRIGGERS FROM `formation`;
30 SELECT *, EVENT_SCHEMA AS `Db`, EVENT_NAME AS `Name` FROM information_schema.`EVENTS` WHERE `EVENT_SCHEMA`='formation';
31 USE `formation`;
32 SELECT * FROM `information_schema`.`COLUMNS` WHERE TABLE_SCHEMA='formation' AND TABLE_NAME='cours' ORDER BY ORDINAL_POSITION;
33 SHOW INDEXES FROM `cours` FROM `formation`;
34 SELECT * FROM information_schema.REFERENTIAL_CONSTRAINTS WHERE CONSTRAINT_SCHEMA='formation' AND TABLE_NAME='cours' AND REFERENCED_TABLE_NAME IS NOT NULL;
35 SELECT * FROM information_schema.KEY_COLUMN_USAGE WHERE TABLE_SCHEMA='formation' AND TABLE_NAME='cours' AND REFERENCED_TABLE_NAME IS NOT NULL;
36 SHOW COLLATION;
37 SHOW ENGINES;
38 SHOW CREATE TABLE `formation`.`cours`;
39 SELECT CONSTRAINT_NAME, CHECK_CLAUSE FROM `information_schema`.`CHECK_CONSTRAINTS` WHERE CONSTRAINT_SCHEMA='formation' AND TABLE_NAME='cours';
Connecté : 00:00 h
MariaDB 10.7.3
Disponibilité : 01:56 h
Heure serveur : 08:
En attente.

INSTALLATION



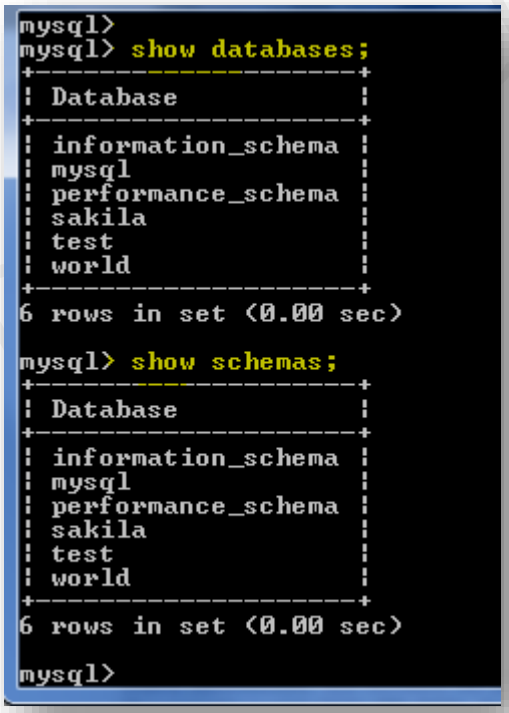


Schémas MySQL MariaDB

Un schéma de base de données schématise la représentation logique d'un **SGBDR** (système de gestion de bases de données relationnelles) **:**

- Il peut représenter un ensemble de contraintes d'intégrités référentielles liant différentes tables.
- Il peut représenter l'ensembles de stables, vues et procédures stockées.

Voici un 1er exemple (forme minimaliste une database)



CONNEXION ET UTILISATION

⇨ **Comment se connecter au serveur MySQL en ligne de commande :**

mysql -u root -p *--Le mot de passe est ensuite demandé …*

use centrale; *-- pour utiliser la base centrale*

mysql -u root -p -D centrale *-- plus directe*

⇨ **Comment créer une de base de données :**

Voici un exemple qui permet de créer la base de données nommée centrale avec un jeu d'encodage de caractères en UTF-8 (***default character set***) et un jeu de règles pour comparer et rechercher les données dans le jeu d'enregistrements (***collate***) basé sur du Case insentive dans notre cas :

create database centrale DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;

⇨ **Comment créer une table en précisant un moteur de stockage :**

```
create table formation (
    id_cours smallint not null auto_increment,
    cours varchar(100),
    temps timestamp default current_timestamp on update current_timestamp,
    primary key(id_cours)
    )
engine = InnoDB; -- default character set utf8mb4 collate utf8mb4_general_ci;
-- le jeu d'encodage est déjà défini dans la base de données parente
```

SCHEMA MYSQL.USER

Le schéma système **mysql** contient les tables qui permettent de gérer totalement les droits des utilisateurs MySQL . Ces tables sont :

- user : contient les privilèges (globaux toutes les bases et toutes les tables) de chaque utilisateur
- db : contient les droits spécifiques à une base (schéma)
- tables_priv : contient les droits spécifiques à une table ou à une vue
- columns_priv : contient les droits spécifiques à une colonne (champ)
- procs_priv : contient les droits des routines (procédures ou fonctions)

```
1   -- 1-Affichage des comptes root
2   SELECT user,host,password FROM mysql.user where user='root';
3
4   -- 2-Suppression des comptes root à distance
5   DROP USER 'root'@'192.168.10.1';
6   DROP USER 'root'@'%';
7   DROP USER 'root'@'127.0.0.1';
8
9   -- 3-Renommer le compte root
10  RENAME USER 'root'@'localhost' TO 'chef'@'localhost';
11  SELECT user,host,password FROM mysql.user where user='chef';
12
13  SELECT user,host,password FROM mysql.user;
14
```

SCHEMA MYSQL.USER

```
use mysql;
show tables;
select * from user; -- \G pour terminer la ligne de commande => affichage mieux formaté
select * from User.user;
select host, user from user;
```

Information_schema


```
-- ------------------------------------------------------------------
-- Administration
-- ------------------------------------------------------------------
use mysql;
show tables;

use information_schema;
show tables;
select * from tables where
    table_schema <> 'mysql'
    and table_schema <> 'information_schema'
    and table_schema <> 'performance_schema'
    and table_schema <> 'sys';

select * from tables where
    table_schema  not in ('mysql','information_schema','performance_schema','sys','sakila');
```

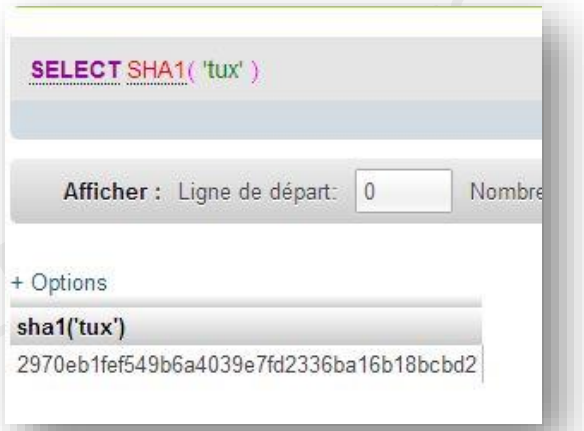
GESTION DES USERS

⇨ Voici un exemple qui permet de créer un utilisateur nommé michel associé au mot de passe tux (clause optionnelle) et qui aura accès au serveur local.

CREATE USER 'michel'@'localhost' IDENTIFIED BY 'tux';

IDENTIFIED encrypte le mdp avec password par défaut

Ou **CREATE USER 'michel'@'localhost';** -- sans mot de passe



⇨ Le mot de passe est ici saisi en clair, il vaudrait mieux utiliser l'empreinte SHA1 ou md5 que l'on peut générer avec la commande select sha1('tux') , ce qui donnerait :

CREATE USER 'michel'@'localhost' IDENTIFIED BY '*AC5CC6D66967DC0DC9222066A7D80A36208DF22D';

Pour rappel md5 et sha1 sont des algorithmes de hashage, ils créent une empreinte du mot de passe et il est donc plus compliqué de récupérer l'original.
Malgré tout les attaques dites de force brute basées sur de puissants dictionnaires peuvent venir à bout d'un hash d'où la nécessité d'utiliser des mots de passe complexes.

⇨ Un autre exemple qui donne les droits classiques uniquement sur la database base_formation et la table table_planning …

GRANT SELECT , INSERT ,UPDATE ,DELETE ,CREATE ,DROP ,ALTER ON base_formation.table_planning TO 'bob'@'localhost';

ALL PRIVILEGES donne tous les privilèges …

SUPER permet de modifier à chaud les variables systèmes …

USAGE ne donne aucun droits … l'utilisateur existe ! FAUX AMI

GRANT USAGE ON * . * TO 'michel'@'localhost';

REVOKE ALL PRIVILEGES ON * . * FROM 'michel'@'localhost'; supprime tous les droits

⇨ Les commandes **GRANT REVOKE** et **SET PASSWORD** sont prises immédiatement en compte par MySQL.
Si on modifie les privilèges d'un utilisateur avec les insert ou update directement dans la table **mysql.user**, on doit exécuter la commande **FLUSH PRIVILEGES** pour prise d'effet.

SET PASSWORD FOR 'bob'@'localhost' = PASSWORD(NouveauMDP');

-- est l'équivalent de :

UPDATE mysql.user SET Password=PASSWORD(NouveauMDP') WHERE User='bob' AND Host='localhost';

FLUSH PRIVILEGES;

GESTION DES USERS

```
-- users et des droits
-- Création d'un utilisateur => son login et son HOST (domaine | @IP )
-- create user 'michel'@'192.168.10.0' ... ou create user 'michel'@'air-france.fr'
drop user if exists 'michel'@'localhost';
create user 'michel'@'localhost' identified by 'michel'; -- identified by 'mot de passe'

-- Attribution des privilèges
-- droits sur toutes les tables d'une base (mysql.db)
grant select, insert, delete on base_formation.* to 'michel'@'localhost';

-- droits globaux (mysql.user)
grant select on *.* to 'michel'@'localhost';

-- droits sur une table précise d'une base (mysql.tables_priv)
grant select, insert, delete on base_formation.formation to 'michel'@'localhost';
-- Révocation des privilèges
revoke all privileges on *.* from 'michel'@'localhost';

-- permet de lister rapidement les privilèges d'un user
show grants for 'michel'@'localhost';
```

GESTION DES USERS

```
-- Rôles
create role admin_niv1;
create role dev;

grant show databases on *.* to dev;
grant select, insert, update , delete on *.* to dev;
set role dev; -- active le role pour la session courante
show grants for dev;

drop user if exists 'tux'@'localhost';
create user 'tux'@'localhost' identified by 'tux'; -- usage === connect !!

grant dev to 'tux'@'localhost';
revoke dev from tux@localhost;

set default role dev for 'tux'@'localhost';
set default role none for 'tux'@'localhost';

show grants for 'tux'@'localhost';

select current_role();
SELECT * FROM information_schema.applicable_roles;
```

TP - USER

```
-- TP
-- 1- Créer un user 'tux'@'localhost'
-- 2- donner un privilège select uniquement sur la colonne cours de la table formation de la base base_formation


-- privilège select et insert au niveau de la colonne cours seulement (mysql.columns_priv)
-- Vérifier les droits
```

TP - USER

- **Créer un schéma nommé world en utf-8**
- **Créer une table ville :**
  - pk_ville (integer auto_increment primary key)
  - Ville varchar(150)
  - Cp varchar(5)
  - Dept varchar(150)
  - Habitants integer
  - Moteur MyISAM
- **Insérer 3 villes dans la table :**
  - Nice 06200 alpes maritimes 343000
  - Sophia Antipolis 06600 Alpes Maritimes 9102
  - Antibes 06600 Alpes Maritimes 75820
- **Importer avec la commande mysql les datas world.sql**
- **Créer 2 utilisateurs : (mdp identique au login)**
  - Bob droits select sur table ville
  - Tux droits select + **execute** sur une base (execute concerne avant tout les bases !!)

```
mysql> grant select, execute on world.* to 'bob'@'localhost';
```

  - Admin tous les droits sur table ville
  - Mac : droits de base (dml) sans mot de passe

LES MOTEURS DE STOCKAGE
MYSQL | MariaDB

INTRODUCTION

Nous allons voir quels sont les principaux moteurs de stockage de données dans le serveur MySQL et quels sont les avantages et les inconvénients de chacun d'entre eux ..
Le rôle du moteur est de stocker les informations sur les disques et en mémoire vive...

MySQL propose une architecture **pluggable storage engine**, ce qui permet de choisir le type de moteur de stockage au niveau de chaque table en fonction des données et des requêtes à traiter.
Le moteur de stockage est choisi à la création de la table mais il peut être modifié plus tard...

⇨ Commençons par voir quels moteurs sont supportés par notre serveur :

```
mysql> select engine, support from information_schema.engines;
+--------------------+---------+
| engine             | support |
+--------------------+---------+
| FEDERATED          | NO      |
| MEMORY             | YES     |
| InnoDB             | DEFAULT |
| PERFORMANCE_SCHEMA | YES     |
| MyISAM             | YES     |
| MRG_MYISAM         | YES     |
| BLACKHOLE          | YES     |
| CSV                | YES     |
| ARCHIVE            | YES     |
+--------------------+---------+
9 rows in set (0,00 sec)
```

⇨ **TP : vérifier quels moteurs de stockage sont gérés par votre serveur …**

QUELS MOTEURS CHOISIR ?

Choisir le moteur de stockage dépend donc de beaucoup de paramètres comme :

- La façon de stocker les données et les index : Le stockage est il physique (disque) ou en mémoire vive ?
- Les sauvegardes et restaurations.
- Les transactions.
- Les index: les full-text, B-tree, hash …

Les index B-TREE sont les index par défaut de MyISAM et innoDB, ils permettent d'effectuer toute une panoplie de comparaison (< <= > >)
Les index FULL-TEXT de MyISAm permettent d'accélérer les requêtes de type filtre sur string
Les index HASH (moteur MEMORY) est beaucoup plus rapide qu'un index B-TREE mais ne permet de faire que des = (WHERE colonne = xxxx)

- Contraintes d'intégrités : les clés étrangères
  Les contraintes peuvent être placées dans l'applicatif ou dans le schéma de la base de données.
- Résistances aux pannes
- Meilleure concurrence : lors d'accès multiples, comment seront gérés les verrous ?
- Rapidité, performances

⇨ **Exemple et comparaison :**

Un moteur transactionnel (innoDB) :

1. Est plus résistant aux pannes
2. Sait gérer une très bonne concurrence

Un moteur non-transactionnel (myIsam) :

1. Peut être plus rapide (en lecture)
2. Occupe moins d'espace disque
3. Consomme moins de mémoire

⇨ **TP : Notez sur le papier quels sont vos critères de choix …**

LE MOTEUR INNODB - XTRADB

INNODB est un moteur transactionnel et relationnel. Il est depuis 2008 soit intégré directement dans MySQL soit livré en tant que Plugin.

Percona XtraDB (pour MariaDB jusqu'à 10.3) est une version améliorée du moteur de stockage InnoDB conçu pour supporter une meilleure montée en charge sur du matériel moderne, il inclus également un grand nombre d'autres fonctionnalités utiles dans les environnements à haute performance.
Il est complètement rétro-compatible et s'identifie à MariaDB en tant que "ENGINE=InnoDB" (tout comme InnoDB) et peux donc être utilisé en remplacement direct de l'InnoDB standard.
(*source : https://mariadb.com/kb/fr/about-xtradb/)*

Il est conforme à la notation **ACID** :

- **Atomicité** : La transaction est complètement acceptée ou totalement refusée
- **Cohérence** : la transaction doit laisser la base dans un état cohérent
- **Isolation** : une transaction ne peut pas interagir avec une autre transaction
- **Durabilité** : une fois acceptée, les résultats de la transaction sont définitivement conservés;

Il est adapté aux tables dynamiques (mises à jour très régulièrement avec beaucoup de transactions)
Il s'assure de la cohérence des enregistrements connexes lors d'opérations sur les tables.

Le verrouillage des données s'effectue uniquement au **niveau de la ligne** concernée et non de la table (MyIsam).

⇨ **innoDB utilise 3 fichiers pour définir ses informations :**

⇨ **ib_logfile0 et ib_logfile1** stockent toutes les transactions *commitées*
Ils permettent de restaurer automatiquement les données près un crash système grâce aux fichiers basé sur les journaux de transactions

Le fichier **.frm** contient la structure et la description de la table. (*ceci est commun à tous les moteurs MySQL jusqu'à la version 5.7).*
Désormais un fichier .sdi indique les méta datas pour MyISAM et les méta-datas sont gérées dans le .ibdata1

⇨ ⇨ Les données et les index de toutes les tables InnoDB du serveur sont stockés dans le **tablespace partagé à toutes les tables** (par défaut **ibdata1**).
On peut modifier le fait de stocker toutes les données dans **ibdata** grâce à la variable globale **:**
⇨ **innodb_file_per_table=1**

Cette option force le serveur MySQL à générer un fichier dont l'extension **.ibd**
qui contient les données et les index de la table.

TP : retrouver des tables Innodb

Nous allons rechercher des informations sur les tables créées dans les différentes base de données et rechercher des tables innoDB:

```
1  show engines;
2  -- affiche les moteurs supportés par le serveur
3  show tables from centrale;
4  -- montre le tables de la base centrale
5  describe produits;
6  -- donne la structure d'une table
7  show table status like 'produits';
8  -- donne des infos sur la table dont le moteur (engine)
9  show create table produits;
10 -- montre la requete SQL de création de cette table
11 select table_schema,table_name, table_type, engine from information_schema.tables where table_type like '%view%';
12 -- recherche les vues dans le serveur de base de données
13 select table_schema,table_name, table_type, engine from information_schema.tables where engine='innoDB';
14 -- recherche toutes les tables dont le moteur est innoDB
15
```

File: Autosize:

table_schema	table_name	table_type	engine
admin_mysql	banque	BASE TABLE	InnoDB
admin_mysql	cart	BASE TABLE	InnoDB
admin_mysql	horloge	BASE TABLE	InnoDB

Une table initialement créé dans le tablespace partagé ibdata1 ne sortira pas automatiquemnt de ce tablespace lorsqu'on modifie la directive **innodb_file_per_table.**

⇨ Il faudra utiliser un **alter table** pour que le serveur crée un fichier .ibd

Exemple : ***alter table pays modify ville varchar(255);***

INNODB

📣**Cas particulier innoDB :**

- le cache d'index et de données pour **innoDB** : **innoDB_buffer_pool_size**

Ce cache mémoire est un des plus importants et les plus gourmands en ressources. Il gère les **INDEX** des tables des bases de données et les datas.

*On peut voir la taille de ces caches grimper jusqu'à **plusieurs Go** voire plusieurs **dizaines de Go** pour le buffer_pool innoDB !*

InnoDB stocke en mémoire aussi bien ses index que ses données .

Plus la taille de **innoDB_buffer_pool_size** est élevée, moins il y aura d'accès disque I/O.

On peut donc conseiller de prendre jusqu'à **80% de la RAM** disponible.

innoDB gère ses propres journaux sur le disque (ib_logfile0) qui stockent toutes les transactions effectuées et qui permettent de restituer les données en cas de rollback par exemple ou de panne.

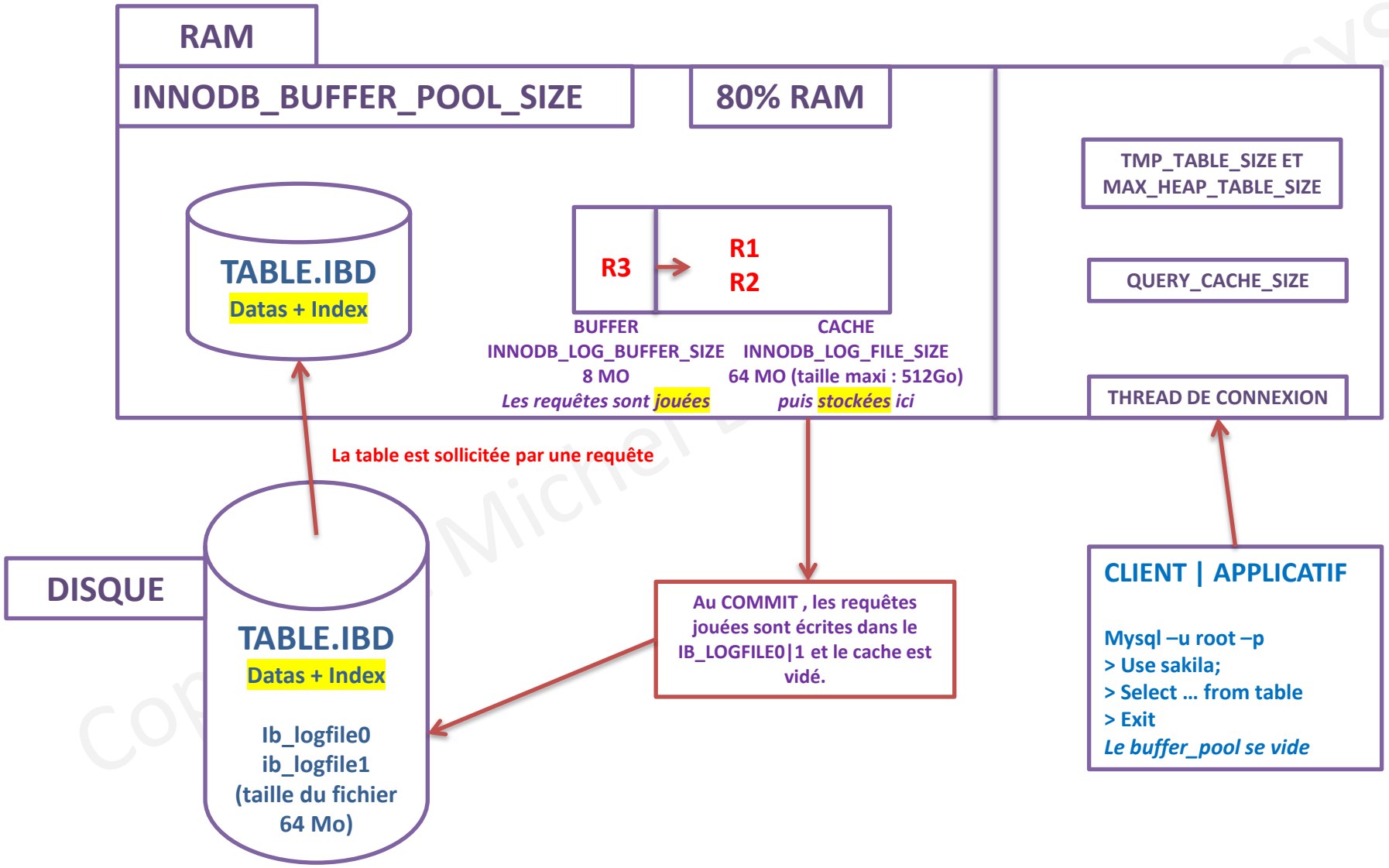
⇨ **innodb_log_file_size** gère la taille de ses journaux (**128Mo est en général une bonne taille) : BUFFER (pile FIFO)**

⇨ **Inno_db_log_buffer_size** gère la taille des buffers des journaux **(de 1 à 8 Mo suffit en général) : CACHE (mém. Permanente)**

⇨ **Optimisation par le COMMIT** pour optimiser les accès I/O **:**
innodb_flush_log_at_trx a pour valeur par défaut ON , chaque transaction est écrite sur le disque à chaque COMMIT (trx validée)

⇨ **Valeur 0 ou OFF** : les écritures se font au minimun 1 fois par seconde (delta max 2700 ?) lors du **checkpoint** (on limite ainsi les accès I/O mais il y a des risques de perte de données lors de panne.

InnoDB


RAM
INNODB_BUFFER_POOL_SIZE
80% RAM
TMP_TABLE_SIZE ET
MAX_HEAP_TABLE_SIZE
QUERY_CACHE_SIZE
THREAD DE CONNEXION
TABLE.IBD
Datas + Index
R3
R1
R2
BUFFER
INNODB_LOG_BUFFER_SIZE
8 MO
Les requêtes sont jouées
CACHE
INNODB_LOG_FILE_SIZE
64 MO (taille maxi : 512Go)
puis stockées ici
La table est sollicitée par une requête
DISQUE
TABLE.IBD
Datas + Index
Ib_logfile0
ib_logfile1
(taille du fichier
64 Mo)
Au COMMIT , les requêtes
jouées sont écrites dans le
IB_LOGFILE0|1 et le cache est
vidé.
CLIENT | APPLICATIF
Mysql –u root –p
> Use sakila;
> Select … from table
> Exit
Le buffer_pool se vide

INNODB

Si la base de données a une taille faible (quelques Go par exemple) et que sa taille doit rester stable dans le temps alors il est facile de choisir une quantité de mémoire vive qui permette à l'intégralité de la base de tenir en mémoire.

Si la base de données a une taille faible mais doit grossir de 1 Go par semaine, il est nécessaire de savoir quelle part de données est utile à un moment donné. Si seules les données de la dernière semaine sont consultées, alors on peut considérer que seul 1 Go de données a besoin d'être en mémoire. Par contre, si 100 % des données sont nécessaires, la base ne pourra pas tenir intégralement en mémoire très longtemps et la question de la performance des disques deviendra vite prépondérante.

Enfin, si la base de données est très volumineuse (1 To, par exemple), la question de la part de données utiles est de nouveau cruciale. Si 10 Go seulement sont réellement utilisés, alors 16 Go de mémoire seront sans doute suffisants, mais si 500 Go sont nécessaires, le choix des disques sera là encore crucial.

Si vous utilisez exclusivement InnoDB, voici un moyen simple pour estimer si vous avez besoin de plus de mémoire. Regardez l'évolution de la variable de **statut** Innodb_buffer_pool_reads lorsque le serveur est chargé :

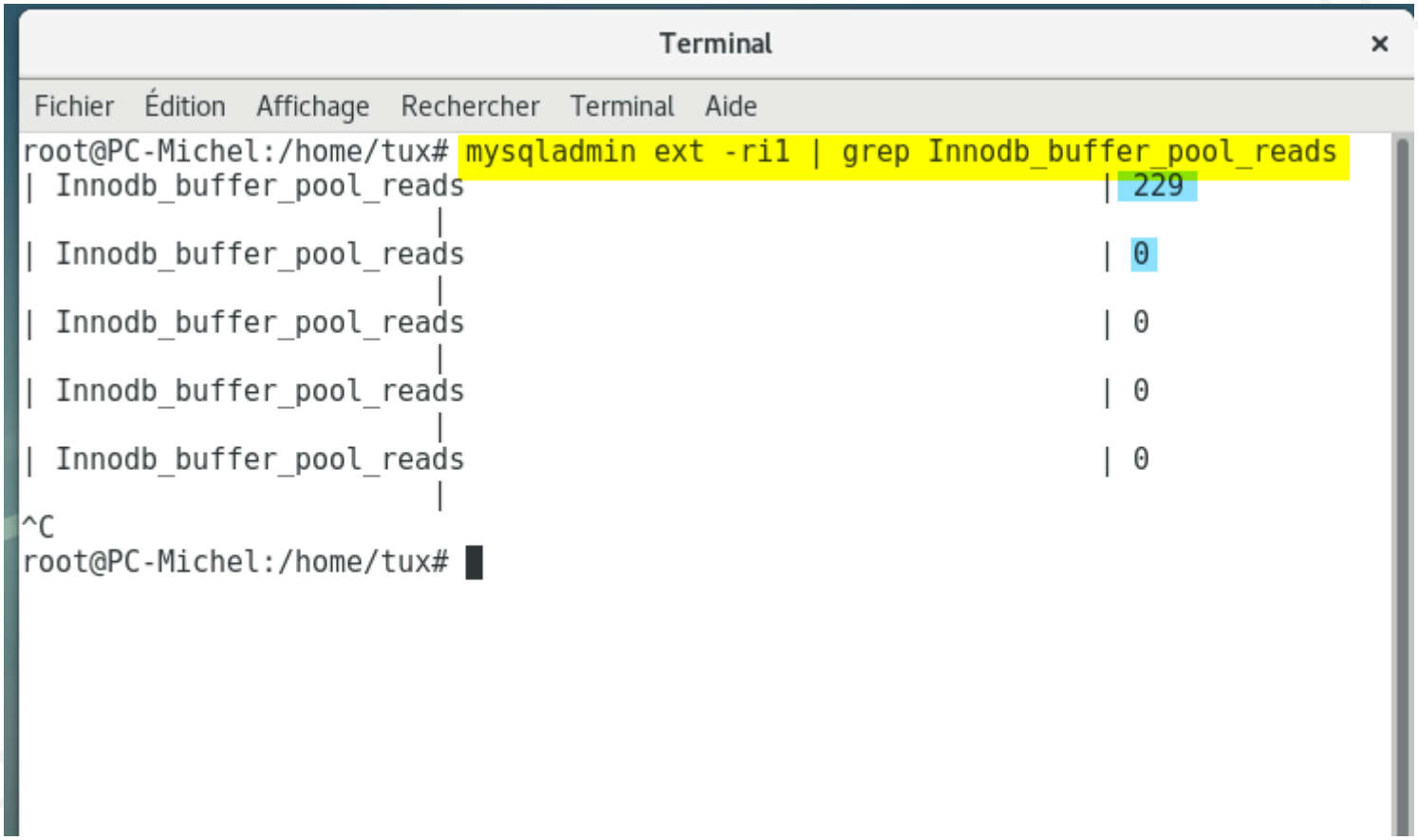
(cf. doc officielle :

> The number of logical reads that `InnoDB` could not satisfy from the buffer pool, and had to read directly from disk.

- si les valeurs sont proches de 0, quasiment toutes les lectures de données sont faites en mémoire
- si les valeurs sont proches du nombre de lectures Innodb_buffer_pool_reads _request … que peut fournir le disque, augmenter la mémoire et la taille du buffer pool va certainement procurer une amélioration des performances.

L'exemple suivant est tiré d'un serveur dont les disques peuvent effectuer environ 2000 opérations/s. La ligne de commande ci-dessous examine la variable Innodb_buffer_pool_reads toutes les secondes (-i1) et affiche seulement la différence par rapport au résultat précédent (-r). Ignorez la première valeur qui donne le nombre de lectures depuis le démarrage du serveur :

INNODB


Terminal
Fichier Édition Affichage Rechercher Terminal Aide
root@PC-Michel:/home/tux# mysqladmin ext -ri1 | grep Innodb_buffer_pool_reads
| Innodb_buffer_pool_reads | 229
| Innodb_buffer_pool_reads | 0
| Innodb_buffer_pool_reads | 0
| Innodb_buffer_pool_reads | 0
| Innodb_buffer_pool_reads | 0
^C
root@PC-Michel:/home/tux#

LE MOTEUR MyISAM

MyISAM était le moteur de stockage par défaut pour toutes les versions de MySQL inférieures à MySQL 5.5
Les tables systèmes utilisent ce moteur. Certaines utilisent également innoDB (versions récentes de MySQL - 8)

☺ Il convient bien aux tables statiques ou pratiquement statiques. (lecture seule !)
☺ Il est simple à gérer … (peu de configuration)
☹ Il ne gère ni les relations avec les contraintes d'inrégrités référentielles, ni les transactions SQL.
☹ Il bloque les tables automatiquement lors d'insertion, de modification ou de suppression de données.
☺ Il permet **l'indexation** des champs et est capable d'optimiser au maximum ses recherches textes avec l'index **FULLTEXT** (bien plus rapide qu'avec une clause LIKE %…)
☺ Il est extrêmement rapide en lecture. (adapté au select)

Les informations d'une table MyISAM sont stockées dans 3 fichiers :

- **nomDeLaTable.frm** : contient la structure de la table (avec MySQL 8 => table.xxx.sdi – serialized dictionnary information)
- **nomDeLatable.MYD** : contient les données
- **nomDeLaTable.MYI :** contient les index

NB: *le fichier .frm est commun à tous les moteurs (<=MySQL 8)*

On peut exécuter plusieurs requêtes en même temps mais la requête en écriture est exclusive et bloque tout le fichier, donc toutes les lignes de la table. De ce fait MyISAM est recommandé pour les tables utilisant beaucoup plus de requêtes en lecture que de requêtes en écriture.

MyISAM : fragmentation et problématique

La fragmentation apparaît lorsque les requêtes effectuent beaucoup de suppression et d'insertion. Il y a alors des trous dans le fichier table. Cela peut poser des problèmes pour la gestion des insertions concurrentes (insertions multiples et simultanées).

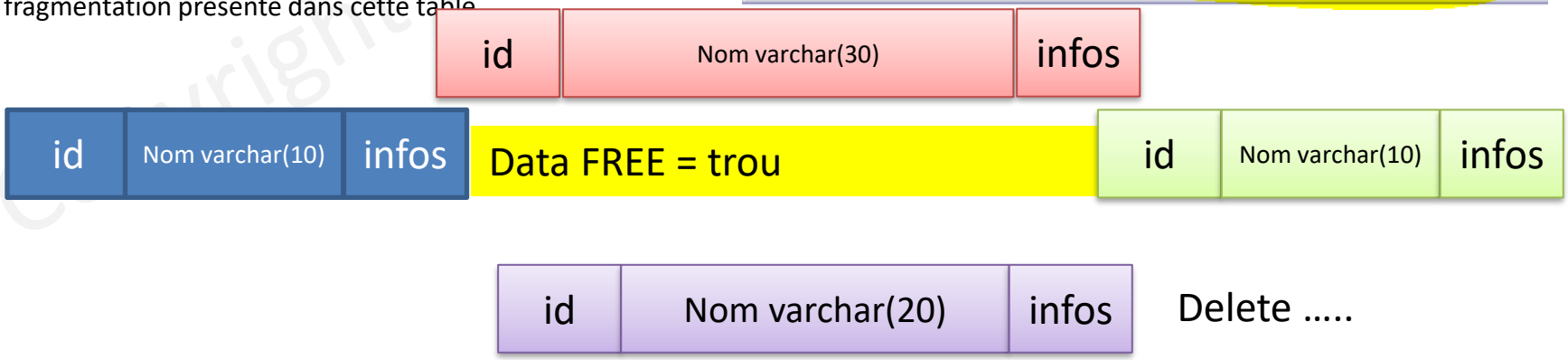
Exemple : si la table n'est pas fragmentée, une insertion peut se faire en fin de fichier pendant qu'une autre requête de lecture est exécutée sur la table. Comment savoir si la table est fragmentée et si elle nécessite une défragmentation avec la commande mysql :

OPTIMIZE TABLE nomDeLaTable …

⇨ *On peut anticiper la défragmentation et les défragmenter régulièrement si l'on sait qu'il y aura beaucoup de delete …*

On peut exécuter cette requête : show table status like 'nomDeLatable' et si data free = 0, alors il n'y a pas de fragmentation présente dans cette table

```
mysql> show table status like 'BIG2'\G
*************************** 1. row ***************************
           Name: BIG2
         Engine: MyISAM
        Version: 10
     Row_format: Dynamic
           Rows: 428480
 Avg_row_length: 85
    Data_length: 100139520
Max_data_length: 281474976710655
   Index_length: 1024
      Data_free: 63315200
 Auto_increment: NULL
    Create_time: 2013-12-13 13:50:39
    Update_time: 2013-12-19 12:38:31
     Check_time: NULL
      Collation: utf8_general_ci
       Checksum: NULL
 Create_options:
        Comment:
1 row in set (0.00 sec)

mysql> delete from BIG2 where Age between 30 and 40;
```



CONCURRENT_INSERT (*fragmentation*)

⇨ Optimisation des insertions avec **concurrent_insert**
(valeur par défaut à 1 = AUTO)

Cette variable système est très importante car elle va conditionner les requêtes de lecture et d'insertion, nous l'avons déjà dit plus haut, si la table n'est pas fragmentée, l'insertion se fait en fin de fichier et les lectures peuvent s'effectuer en même temps.

- Valeur 0 : les Inserts sont **TOUJOURS** verrouillés par les select = toujours prioritaires
- Valeur 1 : (p[illegible]es inserts ne sont **PAS** verro[illegible]ar les selects **SI** il n'y a pas de [illegible]on !!!
- Valeur 2 : les inserts ne sont **JAMAIS** verrouillés par les selects et sont placés automatiquement en fin de fichier.

```
1
2  use admin_mysql;
3  select * from BIG2 limit 0 , 1000000;
4  -- BIG2 contient plus de 1 millions de lignes d'enregistrements
5  -- un simple select prend plus de 5 secondes
6  select count(*) from BIG2;
7  -- renvoie le nombre d'enregistrements
8  show variables like 'concurrent_insert';
9
```

File: Autosize:

Variable_name	Value
concurrent_insert	AUTO

esult 11

tput

tion Output

	Time	Action	Duration / Fetch	Message
1	14:27:07	select * from BIG2 limit 0 , 1000000	0.000 sec / 5.273 sec	1000000 row(s) returned
2	14:30:09	show variables like 'concurrent_insert'	0.000 sec / 0.000 sec	1 row(s) returned
3	14:31:03	select count(*) from BIG2 LIMIT 0, 1000	0.047 sec / 0.000 sec	1 row(s) returned
4	14:31:39	show variables like 'concurrent_insert'	0.000 sec / 0.000 sec	1 row(s) returned

MyISAM – (InnoDB) : format de fichier

» Une table MyISAM peut avoir 3 formats :
On peut à la création imposer le type row_format ou avec alter table

- **statique (FIXED) :**

Ce type de table ne contient aucun champ de type dynamique (VARCHAR, VARBINARY, TEXT ou BLOB.
Tous les enregistrements sont de la même taille, les tables sont plus rapides
La restauration des données est optimisée !
L'espace disque est plus conséquent qu'en table dynamique, c'est le point négatif !
Pas de fragmentation !

- **dynamique (DYNAMIC)**

Les types de données VARCHAR, TEXT et BLOB sont autorisés.
OPTIMIZE TABLE et **myisamchk** sont nécessaires pour défragmenter les tables
La restauration des données est plus contraignante qu'avec les tables statiques.

- ~~**compressée (COMPRESSED)**~~ - *Deprecated*

myisampack permet de compresser les données et les index et de diminuer l'espace disque au maximum de 70 %
UPDATE, DELETE ou INSERT ne fonctionnent plus évidemment.
Il faudra décompresser la table avec **myisamchk**.

→ Une application tournant sur CD/clé USB aura tout intérêt à utiliser une table MyISAM compressée.

```
mysql>
mysql> create table table1
    -> (
    -> id int auto_increment,
    -> nom char(100),
    -> primary key(id)
    -> )
    -> engine=myisam
    -> row_format=fixed
    -> ;
Query OK, 0 rows affected (0.02 sec)

mysql> show table status like 'table1' \G;
*************************** 1. row ***************************
           Name: table1
         Engine: MyISAM
        Version: 10
     Row_format: Fixed
           Rows: 0
 Avg_row_length: 0
    Data_length: 0
Max_data_length: 85849867896750079
   Index_length: 1024
      Data_free: 0
 Auto_increment: 1
    Create_time: 2013-08-16 15:18:01
    Update_time: 2013-08-16 15:18:01
     Check_time: NULL
      Collation: utf8_general_ci
       Checksum: NULL
 Create_options: row_format=FIXED
        Comment:
1 row in set (0.00 sec)

ERROR:
No query specified

mysql>
```

ARIA (MariaDB)

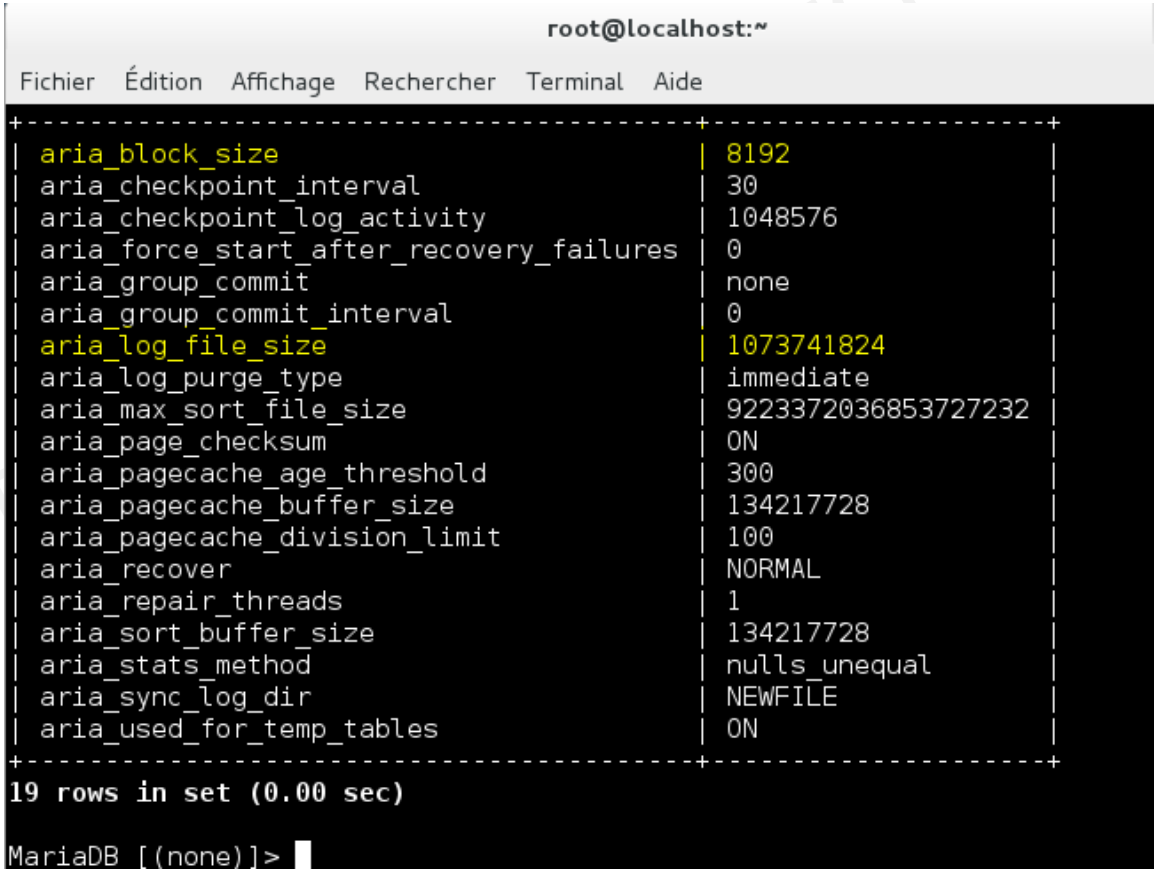
Le moteur Aria tend à remplacer le moteur MyISAM.

C'est l'évolution de MyISAM.

Il est compatible à la norme ACID, est plus résistant aux pannes grâce aux journaux créés aria_log.00000001 et aria_log_control qui en cas de pannes peuvent restituer la table dans un état cohérent (comme InnoDB)

Son fichier de log peut faire 1GO par défaut … la directive aria_log_file_size …

Show variables like '%aria%';



ARIA

Aria gère très bien le concurrent_insert puisque le format par défaut du row_format est Page, un emplacement identique sur le disque dur de 8Ko (que l'on peut augmenter évidemment). ► aria_block_size

Aria gère le row_format static ou dynamic également …

Alter table NomTable row_format=page;

Alter table NomTable row_format=dynamic;

On peut donc faire des requêtes de lecture et d'insert en même temps …

Il n'y a pas de fragmentation non plus grâce au row_format Page …

Pour transformer le moteur de table myISAM en ARIA :

Alter table NomTable engine=aria;

LE MOTEUR MEMORY (HEAP)

Memory est un moteur de stockage qui crée les tables directement en **RAM**.

Il est le plus **rapide** des moteurs mais extrêmement dangereux en cas de plantage du système (les données sont perdues).
Il est utilisé en développement Web pour stocker des informations relatives à la session (Panier, compte client , mail …)
également proposé chez les hébergeurs en mutualisé

Dans ce cas il faut remplir les tables Memory régulièrement depuis une source de données persistante (fichier ou table d'un autre moteur).

Autre cas de figure : MySQL peut être amené à créer des tables temporaires

- cas d'un alter table par exemple ou les données et les index sont totalement vidés de la table existante pour être placés dans une table temporaire le temps de transformer la structure de la table et ensuite y recoller les données et les index
- Cas d'une requête de tri
- Cas d'une requête utilisant les jointures
- Cas d'un group by

Le serveur utilisera une table MEMORY si la taille ne dépasse pas une certaine limite sinon il créera une table MyISAM sur le disque ☹
Comment calculer cette limite ?
tmp_table_size et tmp_memory_table_size (MariaDB) fixe cette limite, en général au moins **512Mo voire 1Go** (*tout dépend de l'applicatif et du système évidement*)

Attention, il existe aussi **max_heap_table_size** (val/défaut : 16Mo) qui fixe la taille maxi des tables crées par le client avec le moteur MEMORY et cette directive aussi va fixer la limite !!

Voila le schéma type, on a optimisé la variable tmp_table_size à 64Mo ou plus, on croit bien faire et comme la valeur par défaut de max_heap_table_size n'est pas modifié dans le fichier my.cnf, dès 16M0 les tables temporaires du système sont écrites sur disque, d'où un écroulement des performances ☹ …. Au moindre group by

LE MOTEUR MERGE

MySQL gère le partitionnement depuis peu.
Il est donc judicieux de découper volontairement les gros fichiers volumineux en partitions …

Auparavant on avait besoin de simuler celui-ci en utilisant des tables de structure identique (MyISAM) et grâce au moteur merge, on pouvait les réunir en une seule table visible …

L'insertion dans une table merge se gère finement avec les options insert_method=first ou insert_method=last. Ce qui fait que les données seront soient insérées dans la 1ère table MyISAM ou la dernière !!

Insert_method=no interdit tout enregistrements depuis la table MERGE !

TP :

- créer 2 tables Janvier2022 et Fevrier2022 de type MyISAM avec les contraintes ci-dessous :

Idlog : int not null auto_increment primary key

log : varchar(255)

- Créer une table merge T2022 avec insert_method=last
- Insérer des données dans cette table T2022
- Vérifier que les données sont écrites dans Fevrier2022
- Inverser ce processus …

Aide à la syntaxe : engine='merge' UNION=(Janvier2022, Fevrier2022)

MERGE : Corrigé

```sql
use admin_mysql;
create table Janvier2013 (idlog int not null auto_increment primary key,log varchar(255) )engine='MyISAM';
create table Fevrier2013 (idlog int not null auto_increment primary key,log varchar(255) )engine='MyISAM';


create table T2013 (idlog int not null auto_increment primary key,log varchar(255) )
engine='merge' UNION=(Janvier2013, Fevrier2013) insert_method = last;
alter table T2013 insert_method=first;

insert into T2013(log) values
('Boumbo;  10-01-2007 10:12:00;  External'),
('Sylvianne; 11-01-2007 14:12:54; External'),
('Albertus;15-01-2007 23:01:20; Internal');


insert into T2013(log) values
('Choulse;  14-02-2007 8:12:56; External'),
('Prospere; 20-02-2007 11:25:17; Internal'),
('Gorgio;20-02-2007 11:26:32; External');

select * from Janvier2013;
select * from Fevrier2013;
```

LES AUTRES MOTEURS

ARCHIVE est un moteur de stockage qui permet de stocker d'énormes quantité de données. en effet, les données sont **compressées** à leur insertion.

Ni les relations, ni les transactions, ni les index ne sont autorisées. on ne peut faire que des requêtes d'insert et de select !
ARCHIVE est principalement utilisé pour **stocker des données brute**s. Un exemple d'application pour archive est **l'enregistrements de logs**.

Le taux de compression peut avoisiner les 70% . Poids fichier super optimisé !
⇒ Peu d'espace disque pour de vieux enregistrements !

CSV : datas au format txt avec séparateur ; , _ … => (Comme certains fichiers Excel)
Compatibilité avec Excel => export de datas déjà présentes dans des tables traditionnelles.

BLACKHOLE : (*source : https://sony-noel.developpez.com/tutorials/mysql/moteurs/?page=blackhole)*

Le moteur BLACKHOLE est l'un des plus mystérieux.
BLACKHOLE, comme en astrophysique, signifie "trou noir". Ce type de table accepte toutes requêtes d'insertion, mais ne renvoie aucun résultat.

« La première utilisation en production serait pour la réplication de données »

Il est possible de mettre un serveur BLACKHOLE comme maître, et d'autres serveurs avec d'autres moteurs comme esclaves. Le moteur BLACKHOLE en tant que maître, acceptera toutes les opérations d'écriture avec de très bonnes performances.

Même si les données ne sont pas stockées, elles seront disponibles dans le fichier de log binaire qui sera envoyé à chaque esclave. Le moteur BLACKHOLE ne sera là que pour relayer les modifications sans supporter les données. Il leur sert de PROXY.

FEDERATED : (depuis la version 5.0.3 de MySQL)

Federated est un moteur de stockage permettant d'accéder à des datas stockées dans des bases de données distantes, et ce sans faire appel à de la réplication ou du clustering.

CASSANDRA SE (Storage Engine) : (depuis la version 10.0 de MariaDB)

Le moteur de stockage Cassandra permet d'accéder aux données d'un cluster Cassandra à partir de MariaDB.
Cassandra SE n'est pas adapté à l'exécution de requêtes de type analytique qui passent au crible d'énormes quantités de données dans un cluster Cassandra.

Cassandra SE est plutôt un « sas » d'un environnement SQL vers NoSQL.

VERROUS …

Les verrous de type **implicite** sont posés par le serveur, plus spécifiquement par le moteur de la table lors **d'insert**, d'update..
Définition de la granularité.

Par défaut, une requête lorsqu'elle est envoyée par le client est mise en attente dans une file jusqu'à ce qu'elle soit exécutée.
Les requêtes de **modification** sont **prioritaires** par rapport aux simples requêtes de **sélection**.

Le système bloque la table ou la ligne grâce aux principes du verrou implicite.
La **granularité** du verrou **implicite** peut être la ligne seulement (moteur InnoDB) ou la table entière (moteur MyISAM).

Le client peut lui interagir avec le système en posant un verrou de type explicite.
Il existe 2 types de verrous explicites (**READ et WRITE**)

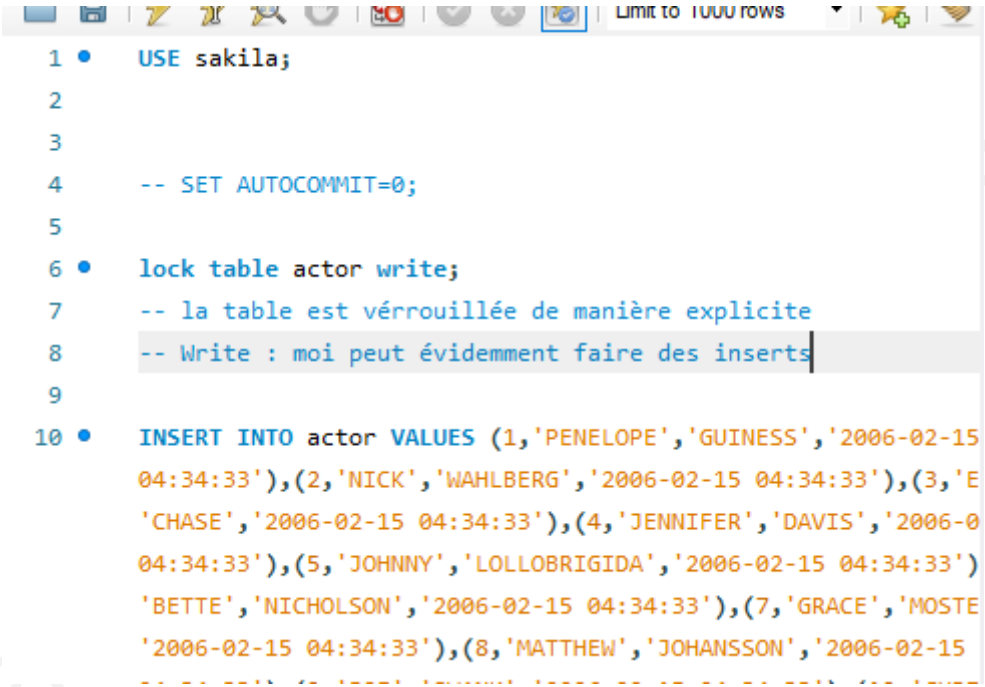
Un verrou posé en lecture seule (READ) bloque la table en modification pour tout le monde y compris le client (moi-même !) qui a posé le verrou :
exemple : **LOCK TABLE MaTable READ;**

Un verrou posé en écriture (WRITE) bloque la table en modification pour tout le monde sauf le client (moi-même!) qui a posé le verrou, lui peut donc modifier la table :
exemple : **LOCK TABLE MaTable WRITE;**

Pour déposer les verrous, il suffit d'exécuter : UNLOCK TABLES;

LOCK TABLES verrouille une table dans le thread courant. UNLOCK TABLES ouvre tous les verrous posé par le thread courant. Toutes les tables verrouillé par un thread sont automatiquement déverrouillée quand le thread émet un autre LOCK TABLES, ou à la fin de la connexion au serveur.
☼ **FLUSH TABLES WITH READ LOCK;** verrouille toutes les tables => idéal pour une sauvegarde

Exemple : Pour voir les tables qui ont actuellement un lock : show open tables
show open tables where in_use > 0 ; -- 1



LES SAUVEGARDES & RESTORATIONS

MYSQL | MariaDB

MYSQLDUMP

MySqlDump est un outil de sauvegarde qui peut exporter et restaurer une ou plusieurs bases de données en incluant les routines, les données et la structure.

Exemple 1: exporte toutes les bases vers svg-25-12-2012.sql
(le user concerné est root avec un mdp debian)

⇨ **mysqldump -u root -p debian –all-databases > svg-25-12-2012.sql**

Exemple 2 : export d'une base de données nommée centrale

⇨ **mysqldump -u root -p debian -B centrale > /tmp/centrale.sql**

Exemple : Importe la sauvegarde au format sql centrale.sql

⇨ **mysql -uroot -pdebian centrale < /tmp/centrale.sql**

Copies et Sauvegardes (MariaDB)

Comment sauvegarder rapidement une base ?

☼ Sauvegarde avec maria-backup : (copie physique)

mariabackup --backup --target-dir=E:\backup --user=root --password=root (backup full)

mariabackup --backup --target-dir=E:\backup-inc-24-11 --incremental-basedir=E:\backup --user=root --password=root (incremental)

mariabackup --backup --target-dir=E:\backup-inc-25-11 --incremental-basedir=E:\backup-inc-24-11 --user=root --password=root

☼ Restauration du backup :

mariabackup --prepare --target-dir=E:\backup (prépare le backup full à ajouter des backup incrémentals)

mariabackup --prepare --target-dir=E:\backup --incremental-dir=E:\backup-inc-25-11 (insère l'incrémental au full)

mariabackup --copy-back --target-dir=E:\backup (copie le backup full + l'incrémental dans le dossier vidé du serveur)

☼ Sauvegarde logique

Alors que mysqldump crée un fichier sql

- Mysqldump –u root –p sakila > /tmp/sakila.sql

- Faire un time pour comparer mais maria-backup est au moins 3 fois plus rapide !

ROUTINES – VUES
TRIGGERS - SCHEDULERS

LES ROUTINES

Les **ROUTINES** sont des **programmes stockés** (procédures stockées ou fonctions) dans le serveur MySQL.

On retrouve les informations les concernant dans la base nommée **information_schema** .

Cette base de données stocke des informations importantes comme :

- Les variables de sessions : **SESSION_VARIABLES** (variables d'optimisation et de tuning : show variables;)
- Le status du serveur : **SESSION_STATUS** (l'état actuel du serveur : show status;)
- Les privilèges des utilisateurs : **USER_PRIVILEGES**
- Les procédures stockées : **ROUTINES**
- Les vues : **VIEWS**
- Les triggers d'événements : **TRIGGERS**
- Et beaucoup d'autres informations …

Les **ROUTINES** permettent d'automatiser des tâches récurrentes d'administration.
Le langage est très simple mais orienté administration..

PROCEDURES STOCKÉES

La **procédure** ne retourne pas en **principe*** de valeur alors qu'une **fonction** retourne elle un résultat.

***** *la procédure peut tout de même retourner un jeu d'enregistrements via une requête SELECT…*

Les banques utilisent les procédures stockées pour toutes les opérations standards.
Cela conduit à un environnement cohérent et sécurisé, car les procédures assurent que les opérations sont correctement faîtes et enregistrées. Dans une telle configuration, les applications et les utilisateurs n'ont aucun accès direct aux tables, mais passent par des procédures stockées pré-définies.

≈ Nous allons prendre un premier exemple avec une procédure stockée :

L'administrateur root crée une procédure stockée nommée **movies** *qui au passage de l'argument* **IN** *cherchera dans les titres des films tout ce qui commence par …*

Toutes les procédures stockées encapsulent leur code par un **DELIMITER** pour éviter à MySQL de chercher à interpréter ces instructions le temps de leur création … Il est d'usage d'utiliser le // ou $$. On pourrait utiliser un simple / …
Les instructions sont également encadrées d'un **BEGIN** et d'un **END**.

EXEMPLES PROCEDURE

Les instruction s sont encapsulées par // ou $$ ou 1 seul /. Par contre il est nécessaire de restituer le bon délimiteur ; comme indiqué en ligne 18.

```
1  use sakila;
2
3  select first_name, last_name, title, description from actor, film, film_actor
4  where actor.actor_id=film_actor.actor_id and film.film_id=film_actor.film_id;
5
6  DROP PROCEDURE if EXISTS movies;
7
8  DELIMITER //
9  CREATE DEFINER=root@localhost PROCEDURE  movies (IN titre VARCHAR(255)) SQL SECURITY DEFINER
10 BEGIN
11 select first_name, last_name, title, description from actor, film, film_actor
12 where actor.actor_id=film_actor.actor_id
13 and film.film_id=film_actor.film_id
14 and title LIKE concat(titre,'%')
15 ;
16 END
17 //
18 delimiter ;
19 call movies('A');
20 use centrale;
21
22
```

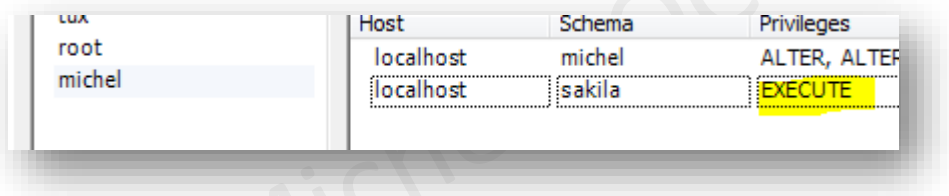
SECURISER UNE PROCEDURE STOCKÉE

On notera également que le propriétaire de la procédure est ROOT (**DEFINER**) et que le **SQL SECURITY** est au niveau **DEFINER**, (au lieu de INVOKER), on verra plus loin que même si l'utilisateur n'a pas de privilèges sur ces tables, il pourra quand même traiter les données.

L'utilisateur michel a uniquement le privilège **execute** sur la base sakila (pour éxécuter les routines)

Il va tout de même essayer de traiter les données de la table actor …

L'accès lui est refusé, par contre s'il utilise la procédure stockée movies, cela fonctionne



L'utilisateur michel a uniquement le privilège **execute** sur la base sakila (pour éxécuter les routines)

Il va tout de même essyaer de traiter les données de la table actor …

L'accès lui est refusé, par contre s'il utilise la procédure stockée movies, cela fonctionne

SECURITE SUITE

```sql
use sakila;

select * from actor;
-- Michel n'a pas le privilège select sur les tables de la base sakila
-- Ce select renvoie une erreur d'accès

call sakila.movies('A');
-- la procédure stockée est appelée avec les droits du créateur soit root,
-- l'accès aux données est autorisé ...
-- Ceci sécurise le tout !
```

Les routines peuvent être sécurisées grâce au definer, il est nécessaire de donner le droit execute aux users qui pourront éxécuter ces routines définies avec le sql security definer

Filter: File: Autosize:

first_name	last_name	title	description
PENELOPE	GUINESS	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies
CHRISTIAN	GABLE	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies
LUCILLE	TRACY	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies

Result 1

Output

Action Output

	Time	Action	Message
1	13:24:43	use sakila	0 row(s) affected
2	13:24:43	select * from actor LIMIT 0, 1000	Error Code: 1142. SELECT command denied to user 'michel'@'localhost' for table 'actor'
3	13:24:48	call sakila.movies('A')	262 row(s) returned

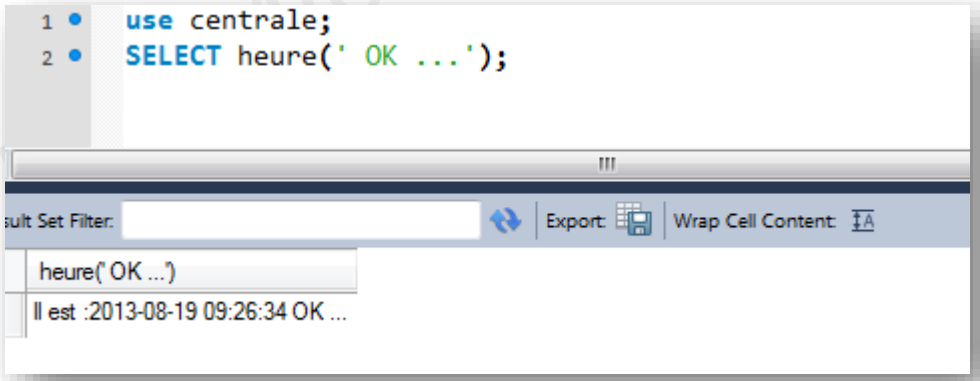
FONCTIONS

La **fonction est une routine qui** peut renvoyer un résultat, elle peut donc faire référence aux méthodes propres de MySQL

```
1  USE `centrale`;
2  DROP function IF EXISTS `heure`;
3
4  DELIMITER $$
5  USE `centrale`$$
6  CREATE FUNCTION heure(message VARCHAR(100)) RETURNS VARCHAR(100)
7  BEGIN
8      DECLARE montre VARCHAR(100);
9      SET montre=NOW();
10     RETURN CONCAT('Il est :', montre, message);
11 END;
12 $$
13
14 DELIMITER ;
```

L'appel de la fonction se fait avec la clause **SELECT** …

```
1  use centrale;
2  SELECT heure(' OK ...');
```



LES VUES

Les **vues** sont des **requêtes stockées** (enregistrées) sur le serveur et retournent des tables de données.

Elles offrent une interface secondaire aux clients et permettent de cacher et rendre non accessible dans une certaine mesure certaines informations de la base.

Elles ne peuvent pas être sécurisées comme les routines mais on peut donner le droit select sur la vue et pas sur le stables concernées par la requête de la vue, ainsi le user n'accède aux colonnes de la table que par le biais de la vue …

```
1  USE `centrale`;
2  CREATE  OR REPLACE VIEW infos_produits AS
3  SELECT produits.nom_p, produits.pu,categories.nom_cat FROM centrale.produits, centrale.categories
4  WHERE produits.id_cat = categories.id_cat;
5  ;
6
```

```
1  SELECT * FROM centrale.infos_produits;
```

sult Set Filter: Export: Wrap Cell Cor

nom_p	pu	nom_cat
Ampli 2x40W JVC	254.25	HI-FI
Platine Disque Philips	45.00	HI-FI
Aspirateur Whirlpool ZX32	95.90	Electroménager
Fer Calor plus Xtra	55.00	Electroménager
Ecran HP 19 pouces	125.38	Informatique
Clavier sans Fil Logitech DX	18.50	Informatique
Jean Bufallo	32.00	Vêtements
Chemise Oxbow	40.00	Vêtements
Livre mon calendrier bota...	12.50	Jardinage

TRIGGERS - DECLENCHEURS

Les triggers permettent d'effectuer certaines actions ou commandes lors d'évènements particulier.(INSERT UPDATE ou DELETE)

```
use base_formation;
drop table if exists t_balance;
create table t_balance (
    id_balance int not null auto_increment,
    montant decimal (7,2),
    primary key(id_balance))
engine ='myisam';
-- ************** Création du trigger -----------------
drop trigger if exists total_montant;
delimiter |
create trigger total_montant before insert on t_balance
for each row
begin
    set @somme=@somme+new.montant;
end
|
delimiter ;
-- ***************** Version plus simple ------
drop trigger if exists total_montant;
-- Création du trigger ------
create trigger total_montant before insert on t_balance
for each row
    set @somme=@somme+new.montant;
-- Initialisation de @somme --------------------------------
set @somme=0;
-- Vidage de t_balance
truncate t_balance;

insert into t_balance (montant) values(150),(124.85),(25.15);
select @somme as 'Total des Montants Saisis';
```

DELIMITER $$
CREATE TRIGGER `NomTrigger`
BEFORE/AFTER INSERT/UPDATE/DELETE
ON `database`.`table`
FOR EACH ROW BEGIN
-- contenu du trigger
-- le code est appliqué à chaque
-- insert/update/delete
END$$

On peut écrire plus simplement le trigger sans le (begin) et le (end) , comme dans l'exemple à gauche

Le before dans le cas d'un insert ne pourra pas récupérer l'ID du nouvel enregistrement !!

SCHEDULER

Il est l'équivalent de CRON sous Linux. il permet d'éxécuter des programmes stockés selon un calendrier.

Voici un exemple simple pour comprendre comment MySQL gère son planificateur de tâches …

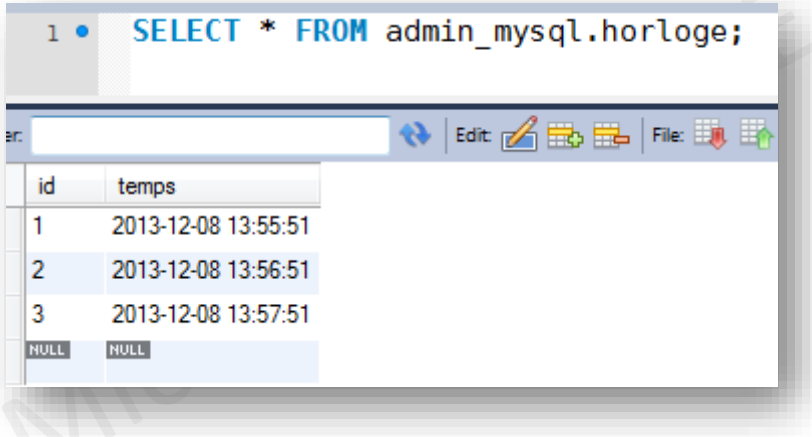
```
-- Mise en fonctionnement du planificateur de taches MySQL
SET GLOBAL event_scheduler = ON;
-- OU
SET GLOBAL event_scheduler = 1;
-- on supprime une table si elle existe
drop table if exists horloge;
create table horloge( id INT AUTO_INCREMENT,temps datetime,PRIMARY KEY (id)) ;
-- Création d'une tache chrono
drop event if exists chrono;

create event chrono on schedule every 1 minute
do
insert into admin_mysql.horloge (temps) values(now());

show events;
```

File: Autosize:

	Name	Definer	Time zone	Type	Execute at	Interval value	Interval field	Starts	Ends	Status	Or
min_mysql	chrono	root@localhost	SYSTEM	RECURRING	NULL	1	MINUTE	2013-12-08 13:55:51	NULL	ENABLED	1
min_mysql	maintenance	root@localhost	SYSTEM	RECURRING	NULL	1	HOUR	2013-08-29 23:51:30	NULL	ENABLED	1

SCHEDULER SUITE …

Et voici le résultat stocké toutes minutes dans la table horloge …



SURVEILLER MYSQL | MARIADB

RAPPELS …

Avant de comprendre comment MySQL gère les allocations mémoires, faisons un bref petit rappel des différences existantes entre les mémoires **registres** | **buffers** | **caches** :
Les registres, buffers et caches sont tous les 3 des zones d'allocations en mémoire.

Leurs différences se situent au niveau de leur gestion et de leur manière d'accéder aux informations des ces zones.

→ Les **registres** (ou mémoire **tampon**) se situent directement dans le processeur et adressent directement les données dans des zones que l'on pourrait comparer à des boites placées exactement là ou elles doivent se trouver. Le registre est adapté aux calculs des demandes des processus.

→ Les **caches** (ou **anté-mémoires**) se situent dans le processeur et/ou en mémoire vive mais la gestion des informations est gérée sous formes de tables de données et les données sont beaucoup plus rapidement accessible et sont permanentes. Le cache pourrait se comparer à un buffer en RAM si ce n'est qu'il conserve les informations.

→ Les **buffers** se situent en RAM dans des files d'attente (exemple : la pile FIFO => First IN First OUT) que gère l'ordonnanceur du noyau du système. Les données sont stockées de manière temporaire.

≠ *MySQL gère de manière particulière la dénomination de ses variables caches et buffers.*
Le **key_buffer_size** par exemple des tables **MyISAM** (décrit ci-dessous) est en fait une mémoire de type **cache** et non un **buffer**…
Il faudra donc veiller à ne pas confondre les définitions générales et le vocabulaire propre à MySQL ≠

VARIABLES SYSTÈMES OU DE STATUT

Les variables de statut permettent de vérifier en temps réel , l'état du serveur :

SHOW GLOBAL STATUS; -- toutes connexions confondues

SHOW SESSION STATUS ou SHOW STATUS; -- de la session uniquement

Les variables systèmes sont définies dans le fichier de configuration de MySQL my.ini ou my.cnf

SHOW GLOBAL VARIABLES;

Pour les étudier, il est plus facile de créer des fichiers que l'on aura tout le temps d'analyser ..

Mysqladmin (comme mysql) est une commande client MYSQL est permet d'effectuer quantité de taches administratives

mysqladmin -u root - p debian variables > /tmp/variables

-- ou

mysql -u root -p -e « SHOW GLOBAL VARAIBLES ; » > /tmp/variables

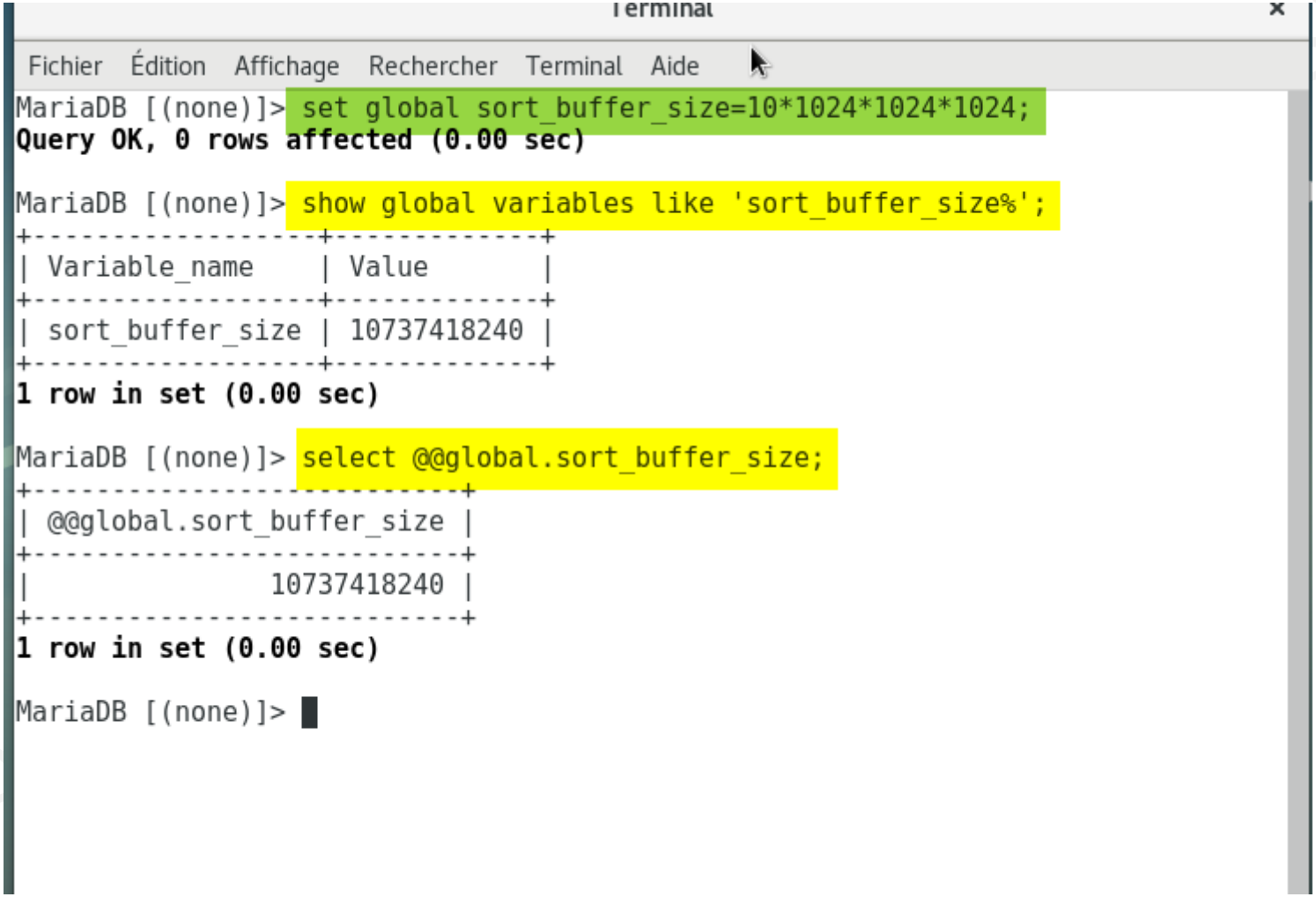
☺ Une variable système peut être modifiée à chaud sans redémarrer MySQL, à condition d'avoir le privilège **SUPER**

VARIABLES SYSTÈMES OU DE STATUT

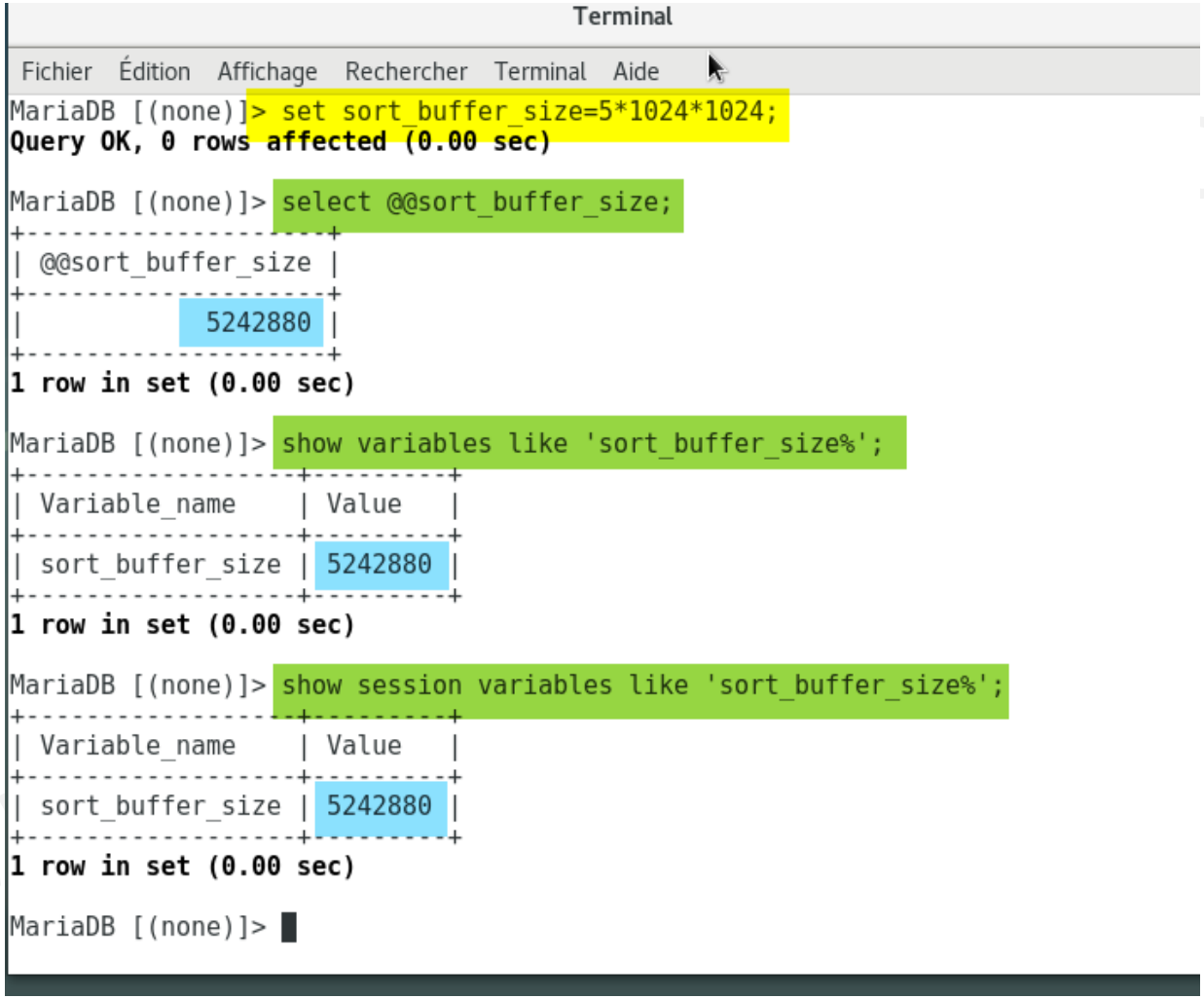
Prenons un exemple simple avec une variable utilisé lors des tris …

```
MariaDB [(none)]> show global variables like 'sort_buffer_size%';
+-------------------+-------------+
| Variable_name     | Value       |
+-------------------+-------------+
| sort_buffer_size | 10737418240 |
+-------------------+-------------+
1 row in set (0.00 sec)

MariaDB [(none)]> show variables like 'sort_buffer_size%';
+-------------------+---------+
| Variable_name     | Value   |
+-------------------+---------+
| sort_buffer_size | 2097152 |
+-------------------+---------+
1 row in set (0.00 sec)

MariaDB [(none)]> show session variables like 'sort_buffer_size%';
+-------------------+---------+
| Variable_name     | Value   |
+-------------------+---------+
| sort_buffer_size | 2097152 |
+-------------------+---------+
1 row in set (0.00 sec)
```

VARIABLES GLOBALES



VARIABLES DE SESSIONS



LE ROLE DU CACHE

Au démarrage du serveur plusieurs allocations mémoires sont crées et utilisées par la suite lors des connexions et requêtes jouées par les clients.

MySQL maintient un ensemble de résultats de requêtes de type SELECT dans une structure en mémoire dédiée appelée **cache de requêtes.**
Lorsqu'un SELECT est exécuté, le serveur stocke le résultat dans le cache de sorte que, si un client demande la même requête dans un temps suffisamment proche pour que les tables n'aient pas changé, le serveur puisse renvoyer directement le résultat qu'il a conservé en mémoire au lieu de passer par toutes les étapes de décomposition, d'optimisation et d'exécution de la requête.

Le cache de requêtes peut donc en théorie considérablement réduire la charge d'un serveur **MariaDB** | **MySQL** en évitant l'exécution de nombreuses requêtes.

SHOW VARIABLES LIKE 'query_cache_type <mysql 8.0.3 -)
Cette variable peut prendre trois valeurs différentes :

ON : le serveur tente de mettre en cache toutes les requêtes **SELECT** sauf celles qui portent le mot-clé **SQL_NO_CACHE** (SELECT SQL_NO_CACHE...).

OFF : aucune requête n'est mise en cache.

DEMAND : le serveur ne tente de mettre en cache que les requêtes **SELECT** comportant le mot-clé **SQL_CACHE** (SELECT SQL_CACHE...).

LE ROLE DU CACHE

UTILITÉ DU CACHE :

Comme indiqué précédemment, le cache de requêtes peut être extrêmement avantageux en termes de performances, mais il existe de nombreux cas où le cache va au final dégrader les performances. En mettant de côté la sérialisation des accès dont il a déjà été question, vous devez aussi considérer les autres coûts du cache de requêtes.

En premier lieu, pour toutes les requêtes SELECT, le serveur vérifie si la requête est présente ou non dans le cache. Cette vérification est bien faite pour toutes les requêtes, même pour celles qui portent un élément les rendant impropres à la mise en cache. Le coût de cette vérification est très faible dans la mesure où il s'agit simplement de chercher si un élément est présent dans une table de hachage en mémoire.

En second lieu, chaque requête SELECT candidate à la mise en cache demande des traitements particuliers. S'il reste suffisamment d'espace dans le cache pour stocker les résultats de la requête, l'opération est rapide. Mais s'il faut effacer des données dans le cache pour récupérer de la place, le traitement peut devenir beaucoup plus coûteux.

Enfin, toutes les requêtes de modification sont ralenties car il faut supprimer toutes les entrées du cache correspondant aux tables modifiées.

On le voit, si l'application subit beaucoup de lectures identiques, le cache va s'avérer très utile. Par contre, le cache de requêtes peut perdre de son intérêt, voire devenir pénalisant si vous êtes dans l'un des cas de figure suivants :

- Les écritures sont nombreuses, provoquant de nombreuses invalidations.
- Les lectures ne sont pas répétées.
- La plupart des lectures ne peuvent pas être mises en cache.

LE ROLE DU CACHE

PARAMÈTRES ASSOCIÉS AU CACHE :

Le cache est paramétré par cinq variables principales :

query_cache_size : il s'agit de la taille du cache. MariaDB alloue toujours une taille multiple de 1024, si bien que la taille réelle du cache n'est pas nécessairement exactement celle que vous aurez demandée. Notez que la totalité de la mémoire définie par ce paramètre est allouée immédiatement.

query_cache_type : indique si le cache est activé ou non. Cette option a été détaillée précédemment.

query_cache_limit : si un résultat de requête a une taille supérieure à query_cache_limit, celui-ci ne sera pas stocké dans le cache.

query_cache_min_res_unit : la mémoire du cache est allouée par blocs dont la taille minimum est donnée par cette variable. Si les résultats à stocker sont très compacts, une valeur trop élevée peut conduire à perdre beaucoup d'espace mémoire.

query_cache_wlock_invalidate : quand un client a verrouillé une table en écriture, un autre client ne peut normalement pas faire de lectures sur cette table. Quand la valeur de cette option est à OFF (valeur par défaut), le serveur peut renvoyer un résultat d'une requête provenant du cache même si la table est verrouillée. Il n'est en général pas nécessaire de modifier cette variable.

```
MariaDB [(none)]> show global variables like 'query%';
+-------------------------------+----------+
| Variable_name                 | Value    |
+-------------------------------+----------+
| query_alloc_block_size        | 16384    |
| query_cache_limit             | 1048576  |
| query_cache_min_res_unit      | 4096     |
| query_cache_size              | 16777216 |
| query_cache_strip_comments    | OFF      |
| query_cache_type              | ON       |
| query_cache_wlock_invalidate  | OFF      |
| query_prealloc_size           | 24576    |
+-------------------------------+----------+
8 rows in set (0.00 sec)
```

Vérifier si les variables de CACHE sont bien configurées ?

Observation des requêtes en cache avec **show global status like '%Qcache%**

- La première variable à examiner est **Qcache_free_memory**, qui indique l'espace libre du cache. Une valeur faible par rapport à la taille du cache peut être le signe d'un cache trop petit, une valeur proche de la taille du cache indique un cache surdimensionné.
- La seconde est **Qcache_lowmem_prunes**, qui donne le nombre d'entrées supprimées dans le cache par manque d'espace. Une valeur élevée est toujours suspecte, et peut être due soit à un cache trop petit, soit à un cache fragmenté.
- La troisième est **Qcache_hits**, correspondant au nombre de fois où le résultat d'une requête a pu être récupéré dans le cache. Si la taille du cache est trop faible, les requêtes insérées dans le cache seront sans doute très vite supprimées par manque d'espace, et peu de résultats pourront être récupérés dans le cache.

Ces trois paramètres doivent vous permettre de trouver une taille de cache en adéquation avec la charge. Sachez que 256 Mo est une taille à essayer de ne pas dépasser. Bien que rien n'empêche en théorie d'adopter une taille plus élevée, en pratique, lorsque le cache dépasse **256 Mo**, les invalidations ou suppressions deviennent longues et provoquent des problèmes de performances.

On y verra :

- la place libre en cache : **Qcache_free_memory**
- Le nombre de requêtes placées en cache : **Qcache_inserts**
- Le nombre de fois ou une requête est ré exploité depuis le cache : **Qcache_hits**
- Le nombre de requêtes non placées en cache : **Qcache_not_cached**

Vérifier si les variables de CACHE sont bien configurées ?

Observation des requêtes en cache avec **show global status like '%Qcache%**

```
MariaDB [(none)]> show global status like '%Qcache%';
+-------------------------+----------+
| Variable_name           | Value    |
+-------------------------+----------+
| Qcache_free_blocks      | 1        |
| Qcache_free_memory      | 16759656 |
| Qcache_hits             | 0        |
| Qcache_inserts          | 0        |
| Qcache_lowmem_prunes    | 0        |
| Qcache_not_cached       | 7        |
| Qcache_queries_in_cache | 0        |
| Qcache_total_blocks     | 1        |
+-------------------------+----------+
8 rows in set (0.00 sec)
```

Vérifier si les variables de CACHE sont bien configurées ?

RÉDUCTION DE LA FRAGMENTATION :

Deux variables vont vous permettre d'estimer la fragmentation du cache, il s'agit de Qcache_free_blocks, qui indique le nombre de blocs libres, et de **Qcache_total_blocks**, qui donne le nombre total de blocs du cache.

Si le rapport **Qcache_free_blocks/Qcache_total_blocks** se rapproche de 1/2, c'est-à-dire qu'entre chaque bloc de données se trouve un bloc libre, votre cache est très fragmenté.

Si votre cache est fragmenté et que la variable **Qcache_lowmem_prunes** a une valeur élevée, alors le serveur est obligé de supprimer des entrées du cache à cause de la fragmentation, ce qui n'est pas souhaitable. Vous pouvez alors défragmenter le cache avec la commande **FLUSH QUERY CACHE** ou changer la valeur de **query_cache_min_res_unit**.

La défragmentation du cache peut prendre du temps, ce qui explique aussi pourquoi il est préférable de dimensionner le cache à **256 Mo** au maximum.

Malgré son nom, la commande **FLUSH QUERY CACHE** ne vide pas le contenu du cache. La commande **RESET QUERY CACHE** se charge de ce vidage.

Afin de trouver une bonne taille pour **query_cache_min_res_unit**, vous pouvez calculer la taille moyenne d'une entrée dans le cache avec la formule (**query_cache_size** - **Qcache_free_memory**)/**Qcache_queries_in_cache**. Si vous avez un mélange de résultats de petite taille et de résultats de grande taille, cette valeur **moyenne** ne sera malheureusement sans doute pas d'une grande aide. Mieux vaut alors par exemple ne pas mettre en cache les résultats de grande taille en choisissant une valeur basse de **query_cache_limit** ou mettre la variable **query_cache_type** à **DEMAND** pour sélectionner quelles requêtes seront mises en cache.

Le CACHE est-il efficace ?

À partir des variables de statut, vous allez pouvoir calculer deux taux intéressants pour déterminer l'efficacité du cache.
Pour ces calculs, vous aurez besoin d'une variable de statut supplémentaire, appelée **Com_select**, et qui donne le nombre de requêtes SELECT qui ont été exécutées par le serveur et non servies par le cache de requêtes :

SHOW GLOBAL STATUS LIKE **'Com_select'**;

Le taux de hits, donné par la formule 100 * **Qcache_hits** + **Com_select**), donne le taux de requêtes servies par le cache par rapport au nombre total de SELECT demandés au serveur.

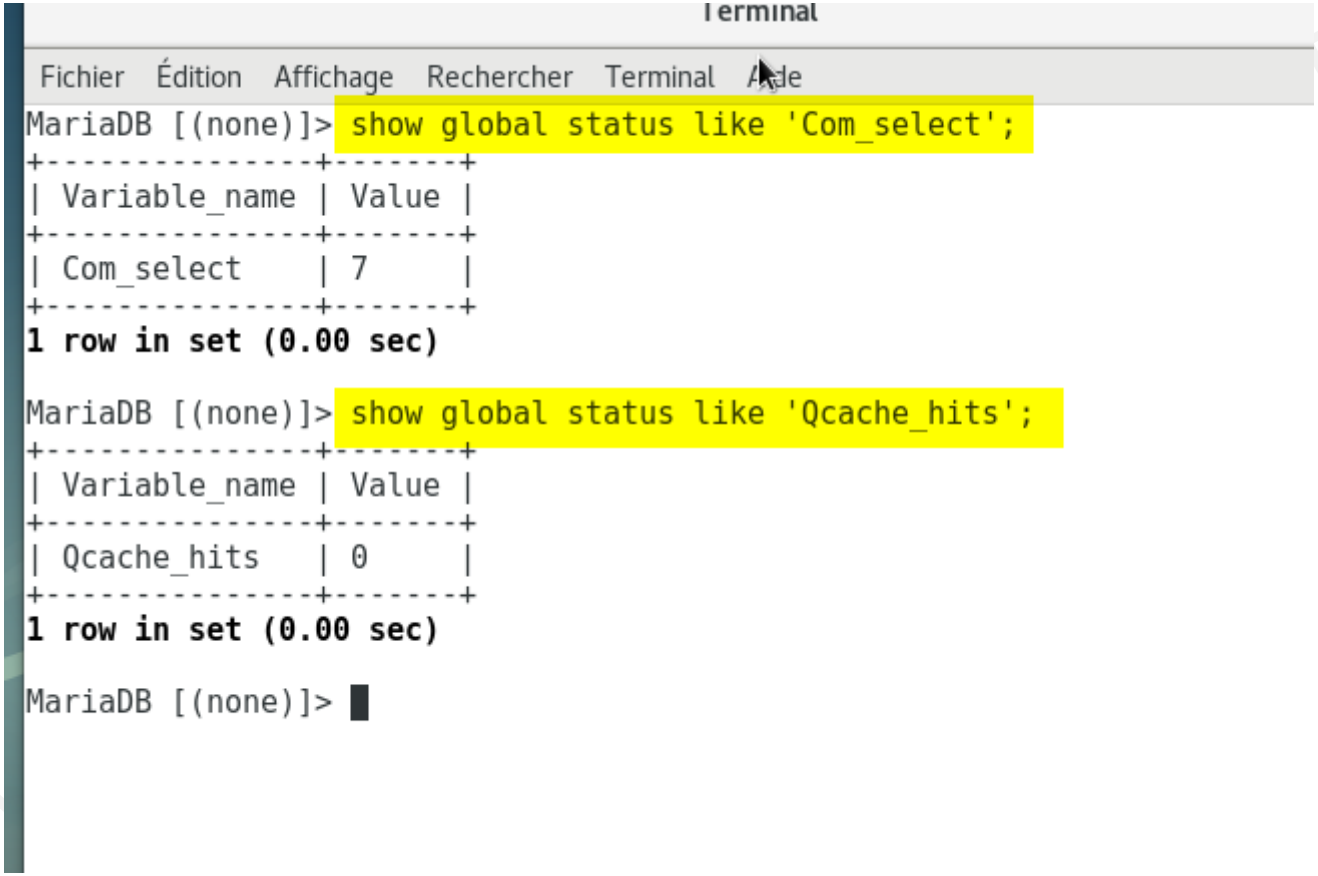
Plus le taux est élevé, plus l'utilisation du cache est intéressante. Cependant, il n'est pas possible de définir de manière absolue ce qu'est un bon ou un mauvais taux de hits.

Dans certains cas, 70 % est parfaitement acceptable alors que dans d'autres il faut viser au moins 99,9 %. Évitez donc de rester complètement focalisé sur l'amélioration de ce taux et ne faites pas confiance aux outils que vous pourrez trouver sur Internet et qui vous indiquent de manière arbitraire si votre taux de hits est bon ou pas.

Le taux d'insertions dans le cache (100 * Qcache_inserts / Com_select) indique la proportion de requêtes SELECT qui sont placées dans le cache lorsque le serveur les exécute.

Si le taux d'insertions et le taux de hits sont élevés, le cache est sans doute efficace et bien configuré.
Si le taux d'insertions et le taux de hits sont faibles, le cache est sans doute peu utile. Vous pouvez alors essayer de le désactiver pour voir si les performances sont meilleures sans cache.
Si les deux taux ont des valeurs moyennes avec l'un élevé et l'autre faible, ces indicateurs ne permettent pas de conclure.
Dans tous les cas, l'information absolue donnée par Qcache_hits et dans une moindre mesure par Qcache_inserts est intéressante à noter.

Le CACHE est-il efficace ?



LES CACHES D'INDEX

LE CACHE D'INDEX :

Le seul réel paramètre important pour les tables MyISAM est la taille du cache d'index, commandée par la variable key_buffer_size.
les données en mémoire sont gérées par l'OS pour MyISAM

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

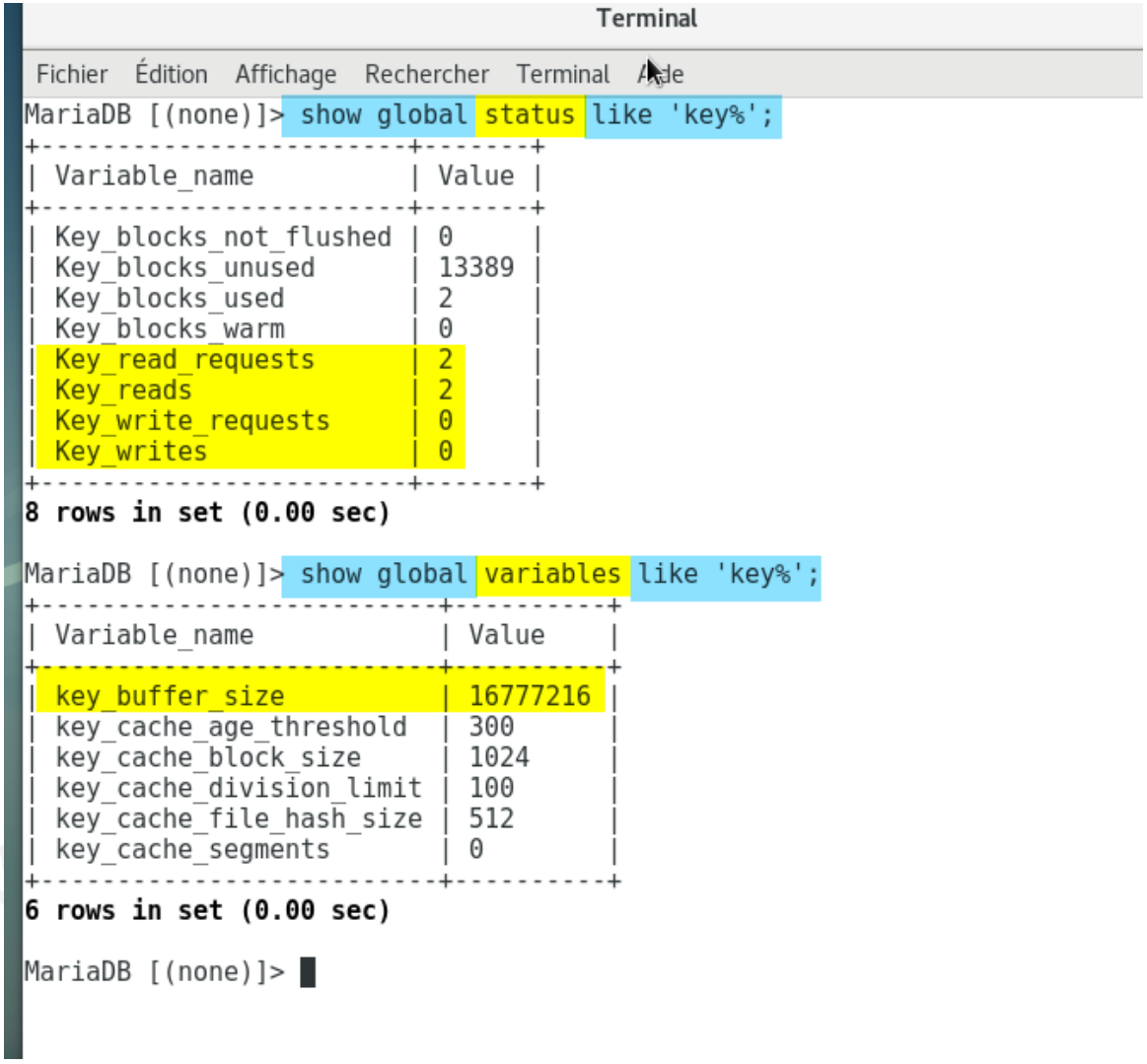
mysql**> SHOW GLOBAL STATUS LIKE 'Key_%';**
Les deux premières variables à considérer sont **Key_reads** et **Key_read_requests**.
Key_reads indique le nombre de requêtes de lectures d'index qui n'ont pas pu être satisfaites par le cache et Key_**read_requests** indique le nombre de lectures faites dans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

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

⇨ Comment optimiser le réglage de la variable key_buffer_size MyISAM ?
Optimisation = 100 – (key_reads / key_read_requests * 100)
Le taux d'échec étant le rapport du nombre de lecture de bloc d'index sur le disque et le nombre de demandes de lecture …

⇨ On pourrait estimer la taille de cette variable entre **30% et 50%** de la **moyenne** de la mémoire disponible pour le moteur **MyISAM**

LES CACHES D'INDEX



CACHE DES REQUETES

LE CACHE DES REQUETES : query_cache_size (*commun aux différents moteurs*) et query_cache_limit

Le résultat des requêtes déjà jouées (le RST) sur le serveur **est stocké dans ce cache**.

Si on note la présence de beaucoup de requêtes récurrentes et rigoureusement identiques, il faut activer le cache des requêtes.

Sa taille maximale conseillée se situe entre **128 et 256 Mo**.

```
mysql> show global variables like '%query%';
+------------------------------+------------------------------+
| Variable_name                | Value                        |
+------------------------------+------------------------------+
| ft_query_expansion_limit     | 20                           |
| have_query_cache             | YES                          |
| long_query_time              | 10.000000                    |
| query_alloc_block_size       | 8192                         |
| query_cache_limit            | 2097152                      |
| query_cache_min_res_unit     | 4096                         |
| query_cache_size             | 33554432                     |
| query_cache_type             | ON                           |
| query_cache_wlock_invalidate | OFF                          |
| query_prealloc_size          | 8192                         |
| slow_query_log               | ON                           |
| slow_query_log_file          | /var/log/mysql/mysql-slow.log |
+------------------------------+------------------------------+
12 rows in set (0.00 sec)
```

Ici la valeur du cache est de 33Mo et la cache est activé.

On en profite pour voir que la journalisation des requêtes lentes est activée et l'on peut noter le chemin des logs …

query_cache_type peut prendre les valeurs OFF et DEMAND:

DEMAND : les requêtes sont stockées dans le cache si elles incluent la clause **SQL_CACHE**

Le cache peut être purgé avec la commande **RESET QUERY CACHE**

CACHE DES TABLES TEMPORAIRES

LE CACHE DES TABLES TEMPORAIRES : tmp_table_size et **max_heap_table_size**

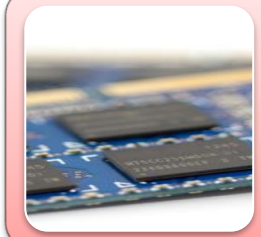
MySQL gère par lui même des tables en mémoire, lors de tri, de group by , d'alter etc ..
La taille maxi des tables temporaires système est fixé par **tmp_table_size**.

Le client peut également créer des tables en utilisant le moteur MEMORY.
La taille maxi de ces tables est fixé par **max_heap_table_size .**

📣 **Attention**, c'est la plus petite valeur de ces 2 variables qui fixe la taille maxi des tables temporaires …
(Déjà vu dans le cours moteur de stockage)

LE CACHE DE TABLE

Le cache de table est commun à tous les moteurs.
Il charge en mémoire la structure des tables (variable **table_definition_cache**) et charge les descripteurs de fichiers (1 fichier = 1table) (variable **table_open_cache**)



RAM – SWAP

table_definition_cache

table_open_cache

DISQUE

Structure des tables (commun à tous les moteurs) => fichier.frm fichier.sdi (stocké dans le tablespace ibdata1)

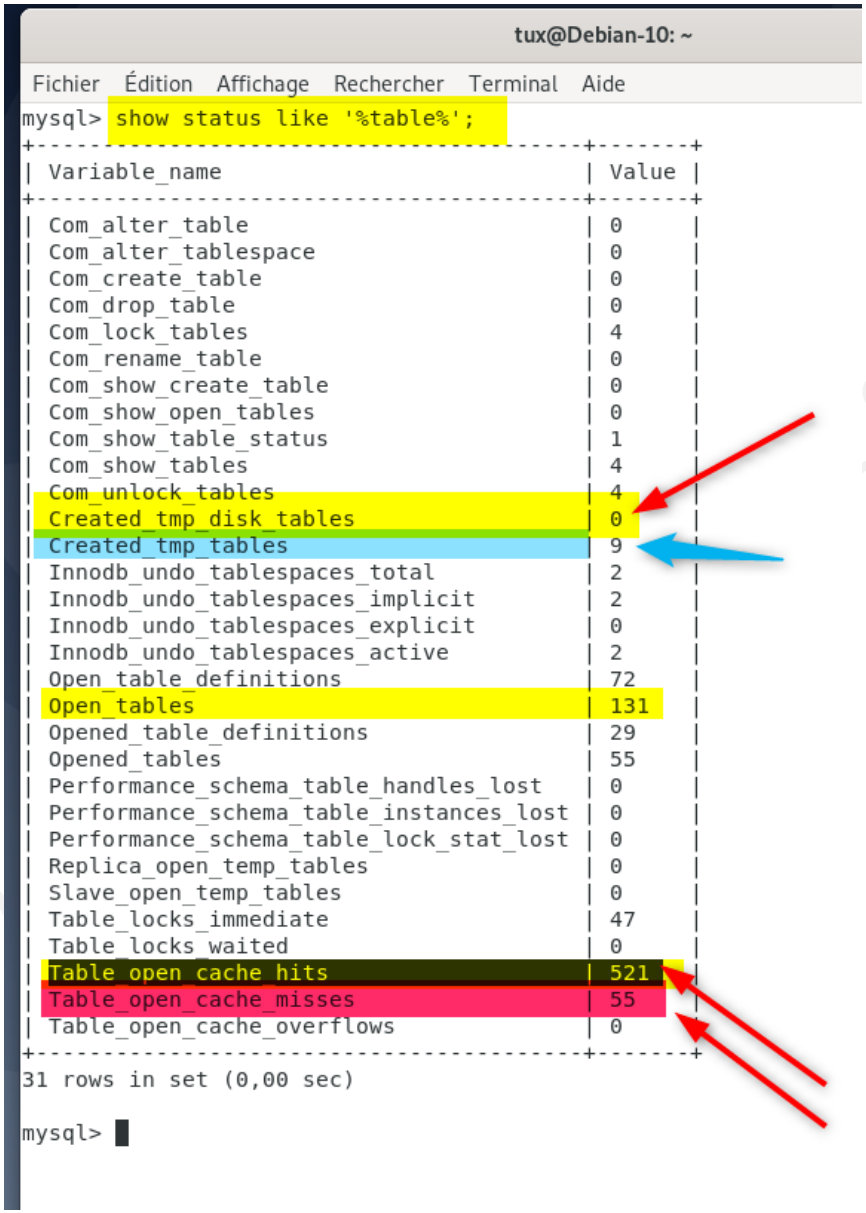
```
mysql> show global variables like 'table_definition_cache';
+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| table_definition_cache | 256   |
+------------------------+-------+
1 row in set (0.00 sec)

mysql> show global variables like 'table_open_cache';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| table_open_cache | 64    |
+------------------+-------+
1 row in set (0.00 sec)

mysql> show global status like '%open_table%';
+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| Com_show_open_tables   | 0     |
| Open_table_definitions | 55    |
| Open_tables            | 49    |
+------------------------+-------+
3 rows in set (0.00 sec)

mysql> show global status like '%opened_table%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| Opened_table_definitions | 47    |
| Opened_tables            | 197   |
+--------------------------+-------+
```

A surveiller (status!)…

- open_table_definitions : nombre de structures de tables en cache
- Open_tables : nombre de descripteur (**tables**) en cache depuis le démarrage du serveur.

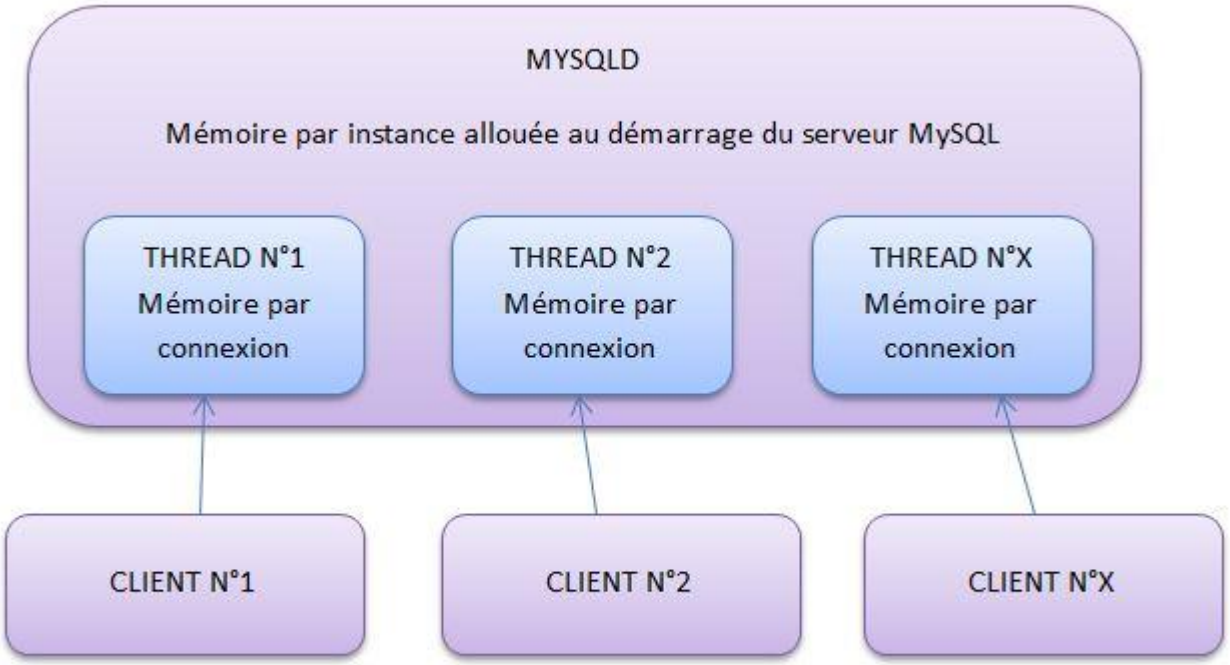
- Opened_tables_definition : nombre total de structures mis en cache
- Opened_tables : nombre total de descripteurs (**tables**) ouverts depuis le démarrage du serveur.

- A faire => une statistique de l'évolution sur opened_tables montrera la moyenne des tables ouvertes par jour.


tux@Debian-10: ~
Fichier Édition Affichage Rechercher Terminal Aide
mysql> show status like '%table%';
Variable_name	Value
Com_alter_table	0
Com_alter_tablespace	0
Com_create_table	0
Com_drop_table	0
Com_lock_tables	4
Com_rename_table	0
Com_show_create_table	0
Com_show_open_tables	0
Com_show_table_status	1
Com_show_tables	4
Com_unlock_tables	4
Created_tmp_disk_tables	0
Created_tmp_tables	9
Innodb_undo_tablespaces_total	2
Innodb_undo_tablespaces_implicit	2
Innodb_undo_tablespaces_explicit	0
Innodb_undo_tablespaces_active	2
Open_table_definitions	72
Open_tables	131
Opened_table_definitions	29
Opened_tables	55
Performance_schema_table_handles_lost	0
Performance_schema_table_instances_lost	0
Performance_schema_table_lock_stat_lost	0
Replica_open_temp_tables	0
Slave_open_temp_tables	0
Table_locks_immediate	47
Table_locks_waited	0
Table_open_cache_hits	521
Table_open_cache_misses	55
Table_open_cache_overflows	0
31 rows in set (0,00 sec)
mysql>

LE CACHE DE THREAD

Chaque client connecté au serveur crée un thread de connexion . Lorsque le client se déconnecte, on peut conserver les informations de connexion dans ce thread et optimiser la prochaine connexion(la variable à personnaliser est **thread_cache_size**)



Le **timeOut** des connexions sans activités est géré par les directives :

- *wait_timeout* (généralement fixé à 28800 secondes)
- *interactive_timeout*

en fonction de comment l'on se connecte).

THREAD et CONNEXIONS … SUITE

A surveiller :

- **threads_cached** : nombre de threads dans le cache
  Pour voir cette variable augmenter, une connection autre que root est faite (mac) et suivie d'un exit. On voit alors le thread_cached passé à 1.

- **threads_connected** : nombre de connexions actuelles
  Exemple : 400 connectés actuellement

- **threads_created** : nombre de threads créés de puis le démarrage du serveur.
  Exemple : 110 000

- **threads_running** : nombre de threads actifs, de threads qui exécutent des requêtes. Dépend de l'activité des connectés.

- **connections** : nombre de connections au serveur
  (abouties ou pas) depuis le démarrage du serveur.
  Exemple : 800 000

- **max_connections** indique le nombre de connexions simultanées. Si **max_used_connexions** dépasse de 1 cette valeur, augmenter le nombre de connexions.

```
mysql> show global status like '%thread%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Delayed_insert_threads  | 0     |
| Slow_launch_threads     | 0     |
| Threads_cached          | 0     |
| Threads_connected       | 2     |
| Threads_created         | 2     |
| Threads_running         | 1     |
+-------------------------+-------+
6 rows in set (0.00 sec)

mysql> show global status like '%connections%';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| Connections          | 44    |
| Max_used_connections | 2     |
+----------------------+-------+
2 rows in set (0.00 sec)

mysql> show global variables like '%connections%';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| max_connections      | 151   |
| max_user_connections | 0     |
+----------------------+-------+
2 rows in set (0.00 sec)
```

Conseils :

threads_created doit augmenter lentement dans le temps
Augmenter le **thread_cache_size** si le rapport *threads_created / connections* reste bas.
Cela veut dire que ce sont les mêmes clients qui se reconnectent régulièrement.

OPTIMISATION DES REQUETES

CACHE DES REQUETES

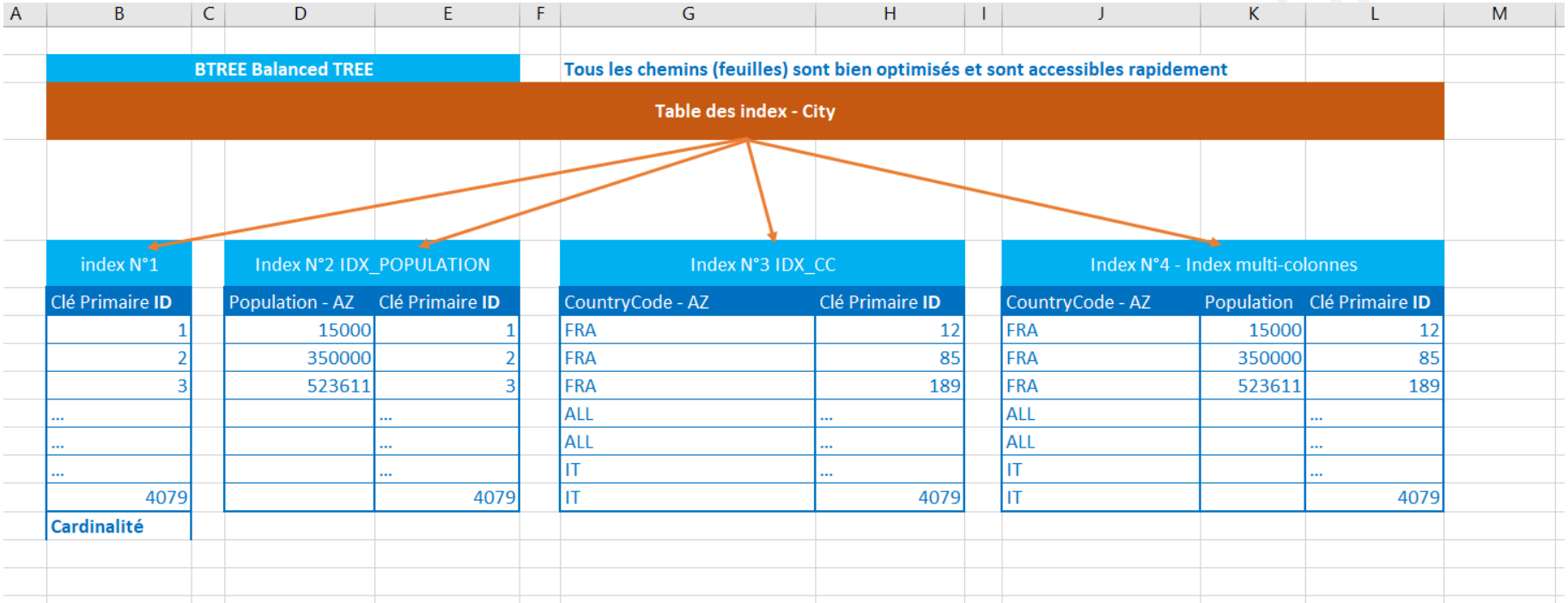
INDEX B-TREE :

Avec MariaDB-MySQL, l'implémentation physique la plus courante d'un index se fait sous la forme d'un **B-Tree** (**Balanced Tree**, soit arbre équilibré en français, ou encore arbre B). Vous obtenez cette information avec la commande SHOW INDEX :

```
mysql> SHOW INDEX FROM world.City\G
....
  Index_type: BTREE
```

Ce type d'arbre est équilibré par nature, ce qui signifie que toutes les feuilles se trouvent à la même distance de la racine. Il a des caractéristiques très intéressantes pour représenter un index : les opérations d'insertion, de suppression et de recherche sont optimisées, les données stockées sont triées et la structure est compacte donc peu coûteuse en espace disque.

INDEX BALANCED TREE



BTREE Balanced TREE

Tous les chemins (feuilles) sont bien optimisés et sont accessibles rapidement

Table des index - City

index N°1	
Clé Primaire **ID**	
1	
2	
3	
...	
...	
...	
4079	
Cardinalité	

Index N°2 IDX_POPULATION	
Population - AZ	Clé Primaire **ID**
15000	1
350000	2
523611	3
	...
	...
	...
	4079

Index N°3 IDX_CC	
CountryCode - AZ	Clé Primaire **ID**
FRA	12
FRA	85
FRA	189
ALL	...
ALL	...
IT	...
IT	4079

Index N°4 - Index multi-colonnes		
CountryCode - AZ	Population	Clé Primaire **ID**
FRA	15000	12
FRA	350000	85
FRA	523611	189
ALL		...
ALL		...
IT		...
IT		4079

CACHE DES REQUETES

LE CACHE DES REQUETES : query_cache_size (*commun aux différents moteurs*)

Les requêtes déjà jouées sur le serveur sont stockées dans ce cache.

Si on note la présence de beaucoup de requêtes récurrentes et rigoureusement identiques, il faut activer le cache des requêtes.

Sa taille maximale conseillée se situe entre **128 et 256 Mo**.

```
mysql> show global variables like '%query%';
+------------------------------+------------------------------+
| Variable_name                | Value                        |
+------------------------------+------------------------------+
| ft_query_expansion_limit     | 20                           |
| have_query_cache             | YES                          |
| long_query_time              | 10.000000                    |
| query_alloc_block_size       | 8192                         |
| query_cache_limit            | 2097152                      |
| query_cache_min_res_unit     | 4096                         |
| query_cache_size             | 33554432                     |
| query_cache_type             | ON                           |
| query_cache_wlock_invalidate | OFF                          |
| query_prealloc_size          | 8192                         |
| slow_query_log               | ON                           |
| slow_query_log_file          | /var/log/mysql/mysql-slow.log |
+------------------------------+------------------------------+
12 rows in set (0.00 sec)
```

Ici la valeur du cache est de 33Mo et la cache est activé.

On en profite pour voir que la journalisation des requêtes lentes est activée et l'on peut noter le chemin des logs …

query_cache_type peut prendre les valeurs OFF et DEMAND:

DEMAND : les requêtes sont stockées dans le cache si elles incluent la clause **SQL_CACHE**

Le cache peut être purgé avec la commande **RESET QUERY CACHE**

INDEX

La commande **show index from NomDeLatable \G** donne les informations des index présents dans la table.

La commande **EXPLAIN** permet de comprendre et d'analyser la requête et voir si elle est optimisée ou pas …

```
mysql> explain select avg(Population) from City group by CountryCode \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: City
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 4079
        Extra: Using temporary; Using filesort
1 row in set (0.00 sec)
```

```
mysql> show index from City\G
*************************** 1. row ***
        Table: City
   Non_unique: 0
     Key_name: PRIMARY
Seq_in_index: 1
  Column_name: ID
    Collation: A
  Cardinality: 4079
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
1 row in set (0.00 sec)

mysql> select count(*) from City;
+----------+
| count(*) |
+----------+
|     4079 |
+----------+
1 row in set (0.04 sec)
```

L'optimiseur fait un full table scan (**type: ALL**) ce qui n'est pas forcement une bonne nouvelle.

Aucun index n'est utilisé (**key: NULL**), ce qui est logique car la table n'en contient pas.

Using temporary; Using filesort indiquent la création d'un table temporaire et le tri des données (pas très bon pour les performances surtout si la table temporaire est créée sur le disque)

Last_query_cost récupère le cout de la requete …

```
mysql> show status like 'last_query_cost';
+-----------------+-------------+
| Variable_name   | Value       |
+-----------------+-------------+
| Last_query_cost | 4963.520924 |
+-----------------+-------------+
1 row in set (0.00 sec)
```

INDEX SUITE

Ajoutons des index :

- ALTER TABLE city ADD INDEX Idx_cc(CountryCode);
- ALTER TABLE city ADD INDEX Idx_population(Population);

- **TP : tester l'explain : y a-t-il une amélioration ?**
- **Si non, effacer les 2 index :**
  ALTER TABLE city DROP INDEX Idx_cc, DROP INDEX Idx_population;

- **La solution : créer un index composite (multi-champs)**
  ALTER TABLE city ADD INDEX Idx_population_cc(Population, CountryCode);

```
mysql> explain select avg(Population) from City group by CountryCode \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: City
         type: index
possible_keys: NULL
          key: idx_population_cc
      key_len: 7
          ref: NULL
         rows: 4079
        Extra: Using index; Using temporary; Using filesort
1 row in set (0.06 sec)
```

INDEX SUITE

Le problème est toujours le même : la requête n'est pas optimisée à 100% …
Elle utilise encore une table temporaire …
Si on analyse les **HANDLERS**, voyons ce que l'on obtient :
Les **HANDLERS** sont des événements de l'interface entre le serveur et les moteurs

```
mysql> show status like '%handler%';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Handler_commit             | 0     |
| Handler_delete             | 0     |
| Handler_discover           | 0     |
| Handler_prepare            | 0     |
| Handler_read_first         | 2     |
| Handler_read_key           | 4079  |
| Handler_read_next          | 8158  |
| Handler_read_prev          | 0     |
| Handler_read_rnd           | 232   |
| Handler_read_rnd_next      | 28814 |
| Handler_rollback           | 0     |
| Handler_savepoint          | 0     |
| Handler_savepoint_rollback | 0     |
| Handler_update             | 3847  |
| Handler_write              | 28800 |
+----------------------------+-------+
15 rows in set (0.00 sec)
```

HANDLER_READ_FIRST comptabilise le nombre de fois que la première valeur de l'index est lue.

HANDLER_READ_KEY indique le nombre d'enregistrement récupéré grace à l'index.

HANDLER_READ_NEXT indique une lecture ordonnée de l'index (une valeur, puis la suivante, puis la suivante…).

Handler_read_first, Handler_read_key & Handler_read_next, indiquent là, un full index scan

Handler_read_rnd & Handler_read_rnd_next indiquent un full table scan sur la table temporaire

HANDLER_UPDATE nous donne une indication sur le nombre de mise à jours dans la table temporaire (à cause du tri)
HANDLER_WRITE indique le nombre de lignes insérées dans la table temporaire

Ces 2 derniers paramètres confirme donc la **création de la table temporaire et l'opération de tri**

OPTIMISATION DES INDEX

Comment faire pour obtenir ceci ?

- Pas de création de table temporaire
- Pas de full table scan sur la table temporaire mais un full scan index …

- Essayer aussi : show status like '%sort%'; et show status like '%created%';

```
mysql> show status like '%handler%';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Handler_commit             | 0     |
| Handler_delete             | 0     |
| Handler_discover           | 0     |
| Handler_prepare            | 0     |
| Handler_read_first         | 1     |
| Handler_read_key           | 0     |
| Handler_read_next          | 4079  |
| Handler_read_prev          | 0     |
| Handler_read_rnd           | 0     |
| Handler_read_rnd_next      | 20421 |
| Handler_rollback           | 0     |
| Handler_savepoint          | 0     |
| Handler_savepoint_rollback | 0     |
| Handler_update             | 0     |
```

```
mysql> explain select avg(Population) from City group by CountryCode \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: City
         type: index
possible_keys: NULL
          key: idx_population_cc
      key_len: 7
          ref: NULL
         rows: 4079
        Extra: Using index
1 row in set (0.02 sec)
```

ECRITURE DES JOINTURES

```
mysql>
mysql> explain select a.first_name, a.last_name , f.title from actor a inner join film_act
or fa using (actor_id) inner join film f using (film_id) order by f.title limit 10 \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: f
         type: index
possible_keys: PRIMARY
          key: idx_title
      key_len: 767
          ref: NULL
         rows: 5
        Extra: Using index
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: fa
         type: ref
possible_keys: PRIMARY,idx_fk_film_id
          key: idx_fk_film_id
      key_len: 2
          ref: sakila.f.film_id
         rows: 2
```

Priorité des requêtes

Afin d'optimiser les requêtes MySQL, le client peut donner un ordre de priorité à l'exécution des ses requêtes

INSERT DELAYED into MaTable (col1,col2) VALUES (x,y);

DELAYED pour les requêtes de type INSERT ou REPLACE redonne instantanément la main au client et place la requête dans un tampon mémoire qui n'exécutera les requêtes que lorsque toutes les autres requêtes 'normales' auront été exécutées.

INSERT LOW_PRIORITY into MaTable (col1,col2) VALUES (x,y);

LOW_PRIORITY ne fonctionne que pour les requêtes de type INSERT | UPDATE | REPLACE | DELETE et pas pour le SELECT.
Les requêtes de modification perdent leur priorité naturelle …

SELECT HIGH_PRIORITY * FROM MaTable;

HIGH_PRIORITY fonctionne également avec le SELECT.
Les requêtes gagnent en priorité …

LE PARTITIONNEMENT MYSQL | MARIADB

La répartition de charge nous permet de gérer la « scalabilité » des données (ces données augmentent avec le temps).

1. Scalabilité verticale :

    - Augmenter les capacités d'une machine (serveur) :
        - CPU (mémoire + proc)
        - Unités de disque (stockage)
    - On peut vite atteindre une limite !

2. Scalabilité horizontale : sharding ou le partitionnement (sur plusieurs machines optimisées..)

    1. On n'a plus de limites en terme d'extensions (autant de nouvelles datas)
    2. Chacun des nœuds du partitionnement doit être capable de fournir les datas quoiqu'il arrive :
    (Haute disponibilité : réplication ou du clustering)

LE PARTITIONNEMENT MYSQL | MARIADB

Le **partitionnement** apporte plusieurs avantages à un administrateur de base de données.
Voici les principaux intérêts du partitionnement :

Pouvoir créer des tables plus grandes que la taille permise par un disque dur ou par une partition du système de fichiers : il est tout à fait possible de stocker des partitions à des endroits (partitions, disques, serveurs, ...) différents.

Pouvoir supprimer très rapidement des données qui ne sont plus utiles et utilisées : si ces données sont placées sur une partition séparée, il suffit de détruire la partition pour supprimer toutes les données.

Optimiser grandement certaines requêtes : les données étant organisées dans différentes partitions, le SGBD n'accède qu'aux données nécessaires lors des requêtes. Sans partitionnement, tous les enregistrements sont pris en compte.

Ce processus peut permettre une gestion plus efficace des données stockées en les regroupant dans différents emplacements (partitions) selon des critères bien définis .

Le partitionnement d'une table se définit à sa création, en indiquant le **type** et la **clé** de partitionnement, ainsi que d'autres **paramètres** tels que le nombre de partitions à générer.

Plus une table est sollicitée , plus le risque qu'elle pose des problèmes de performances augmente. Le partitionnement peut permettre de réduire la **contention** d'une table en la répartissant sur l'ensemble des partitions, grâce au « **partition pruning** ».

DEFINITION

La fonctionnalité de partitionnement **MYSQL** | **MARIADB** permet aux développeurs et aux DBA d'améliorer les performances des bases de données et de simplifier la gestion de bases de données de très grande ampleur.

MySQL autorise un partitionnement horizontal dans lequel les lignes d'une base de données sont divisées en plus petits ensembles de données, puis distribuées dans plusieurs répertoires et sur plusieurs disques.

Le partitionnement permet d'isoler les enregistrements atteints. Cela permet de diminuer la taille des Index et par incidence diminuer leur stockage dans le cache !

Le partitionnement augmente les performances des requêtes puisque des ensembles de données plus petits ont uniquement besoin d'un accès à des opérations spécifiques, et non à une table unique de grande taille. Il est également possible de répartir une table partitionnée sur des lecteurs physiques différents, ce qui permet de réduire la contention sur les interfaces d'E/S physiques en cas d'accès simultané à plusieurs partitions.

Le partitionnement simplifie aussi la gestion des données.

Par exemple, sans avoir à intervenir personnellement, un DBA peut supprimer des partitions spécifiques dans une table partitionnée tout en conservant intactes les partitions restantes (plutôt que d'élaborer une opération de suppression en masse, génératrice de fragmentation, dans la table entière).

PARTITION PRUNING

Le « **partition pruning** » permet à l'optimiseur de ne pas analyser les partitions qui ne contiennent pas de données concernées par l'éxécution de la requête !!

Nativement dans le plan d'exécution des requêtes !

PARTITIONNEMENT BY RANGE

Partionner par rangées va permettre de créer plusieurs petits fichiers et éviter d'avoir de gros volumes de données dans le même fichier :

Les fichiers *.MYD et *.MYI seront dupliqués avec des # . Le traitement sera le même pour les fichiers *.ibd.

```
mysql> alter table BIG2 partition by range(Age) ( partition p_id_moins40 VALUES LESS THAN (40), partition p_id
_moins60 VALUES LESS THAN (60), partition p_id_plus60 VALUES LESS THAN (MAXVALUE) )\G
Query OK, 428480 rows affected (21.11 sec)
Enregistrements: 428480  Doublons: 0  Avertissements: 0

mysql> explain partitions select * from BIG2\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: BIG2
   partitions: p_id_moins40,p_id_moins60,p_id_plus60
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 428133
        Extra:
1 row in set (0.00 sec)
```

```
[root@]$ ls -lh BIG2*
-rw-rw---- 1 mysql mysql 8,7K 19 déc.  13:11 BIG2.frm
-rw-rw---- 1 mysql mysql   60 19 déc.  13:11 BIG2.par
-rw-rw---- 1 mysql mysql  28M 19 déc.  13:12 BIG2#P#p_id_moins40.ibd
-rw-rw---- 1 mysql mysql  36M 19 déc.  13:12 BIG2#P#p_id_moins60.ibd
-rw-rw---- 1 mysql mysql  10M 19 déc.  13:12 BIG2#P#p_id_plus60.ibd
[root@]$
```

SUITE BY RANGE

Exemple avec un fichier MyISAM

```
[root@]$ ls -lh BIG2*
-rw-rw---- 1 mysql mysql 8,7K 19 déc.  13:17 BIG2.frm
-rw-rw---- 1 mysql mysql   60 19 déc.  13:17 BIG2.par
-rw-rw---- 1 mysql mysql  15M 19 déc.  13:17 BIG2#P#p_id_moins40.MYD
-rw-rw---- 1 mysql mysql 1,0K 19 déc.  13:17 BIG2#P#p_id_moins40.MYI
-rw-rw---- 1 mysql mysql  19M 19 déc.  13:17 BIG2#P#p_id_moins60.MYD
-rw-rw---- 1 mysql mysql 1,0K 19 déc.  13:17 BIG2#P#p_id_moins60.MYI
-rw-rw---- 1 mysql mysql 1,4M 19 déc.  13:17 BIG2#P#p_id_plus60.MYD
-rw-rw---- 1 mysql mysql 1,0K 19 déc.  13:17 BIG2#P#p_id_plus60.MYI
[root@]$
```

SUPPRIMER LE PARTITIONING

Terminal

```
mysql> alter table BIG2 remove partitioning;
Query OK, 428480 rows affected (1.93 sec)
Enregistrements: 428480  Doublons: 0  Avertissements: 0

mysql>
```

Terminal

```
[root@]$ ls -lh BIG2*
-rw-rw---- 1 mysql mysql 8,7K 19 déc.  13:22 BIG2.frm
-rw-rw---- 1 mysql mysql  36M 19 déc.  13:22 BIG2.MYD
-rw-rw---- 1 mysql mysql 1,0K 19 déc.  13:22 BIG2.MYI
[root@]$
```

BY HASH

Le partitionnement BY HASH permet de pouvoir créer autant de partitions que l'on veut et permet de répartir plus facilement les données.

By hash sur clé primaire répartira de manière égale les enregistrements.

```
mysql> alter table BIG2 partition by hash  (Age) partitions 20 ;
Query OK, 237120 rows affected (26.76 sec)
Enregistrements: 237120  Doublons: 0  Avertissements: 0

mysql>
```

```
[root@]$ cd /var/lib/mysql/sakila
[root@]$ ls -s BIG2*
   12 BIG2.frm           96 BIG2#P#p15.ibd  11268 BIG2#P#p4.ibd
    4 BIG2.par           96 BIG2#P#p16.ibd  11268 BIG2#P#p5.ibd
   96 BIG2#P#p0.ibd     512 BIG2#P#p17.ibd  10244 BIG2#P#p6.ibd
 9220 BIG2#P#p10.ibd     96 BIG2#P#p18.ibd  11268 BIG2#P#p7.ibd
   96 BIG2#P#p11.ibd     96 BIG2#P#p19.ibd  11268 BIG2#P#p8.ibd
 9220 BIG2#P#p12.ibd  10244 BIG2#P#p1.ibd   10244 BIG2#P#p9.ibd
  564 BIG2#P#p13.ibd  11268 BIG2#P#p2.ibd
   96 BIG2#P#p14.ibd   9220 BIG2#P#p3.ibd
[root@]$
```

REPLICATION MYSQL | MariaDB

La réplication de données consiste à « copier » **le contenu d'une base de données source vers une base de données cible**.

Une réplication complète est un processus qui permet d'avoir des données cohérentes, redondantes afin d'assurer à la fois la performance et renforcer la fiabilité de l'architecture de données.

Les utilisateurs peuvent alors avoir accès à des jeux de données synchronisés avec un temps de latence variable en fonction du contenu de réplication.

La réplication est une fonctionnalité native dans MySQL. Elle est relativement simple à mettre en œuvre dans une configuration basique et procure une grande stabilité dans le traitement des données répliquées.

La réplication MySQL peut alors être utilisée dans les cas de haute disponibilité, de performance, de répartition de charge, de tolérance de panne.

La réplication MySQL offre une solution globale de performance.

Sur un environnement de bases de données répliquées, seules les modifications seront propagées sur l'environnement esclave.

En mettant en place une stratégie de répartition de charge, les accès lectures étant plus nombreux dans une configuration courante, le coût d'activité au niveau de l'environnement esclave sera moins important.

Enfin la sauvegarde en ligne est un avantage supplémentaire. Elle peut représenter une forte importance stratégique.

Les sauvegardes pourront être lancées sur l'esclave sans bloquer les données à la source

REPLICATION UNI DIRECTIONELLE

La réplication classique UniDirectionnelle:

Les bases de données du serveur Maître (N°1) sont recopiées en temps réel vers le serveur Esclave (N°2)

Comment se gère un tel modèle :

Cas N°1 :

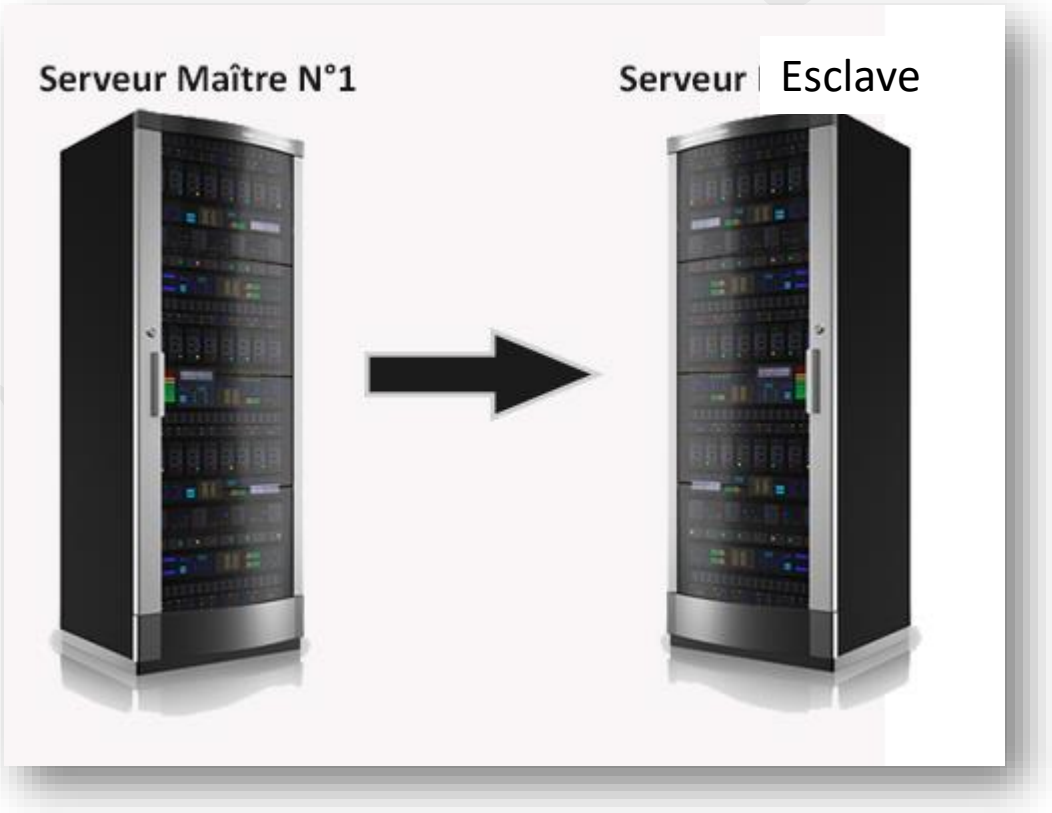
Les requêtes d'écriture se font sur le serveur N°1
Les requêtes de lecture se font sur le serveur N°2
Ce qui permet de répartir les charges **en lecture et en écriture**

Cas N°2 :

Le serveur N°1 est considéré comme le serveur en production
Le serveur N°2 est considéré comme un **Backup Server**

Cas N°3 :

Mix des 2…



REPLICATION CROISEE

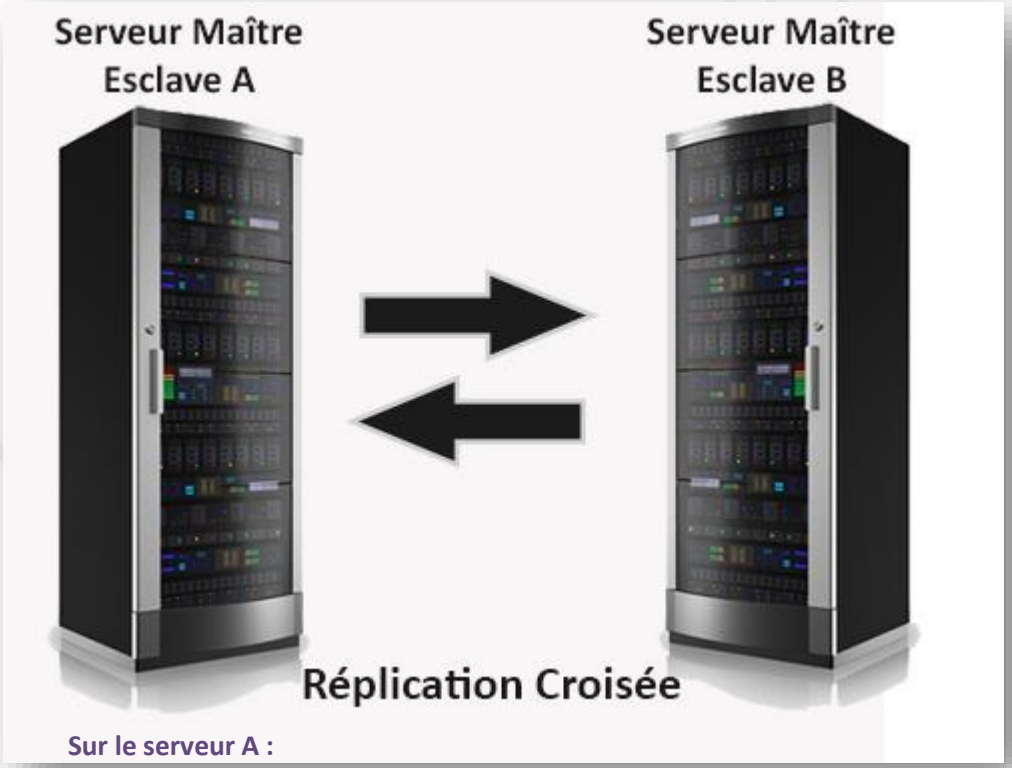
Dans ce cas de figure, les deux serveurs sont à la fois Maître et Esclave et la synchronisation se fait dans les deux sens;
Optimisation de la charge et Sécurisation des données !

CONFLITS D'INSERTION

Si il y a des conflits d'insert avec des clés identiques (**1062**. « DUPLICATE ENTRY FOR…) , la réplication va s'arrêter net et l'on risque de perdre beaucoup d'informations de sauvagrde.

Si le serveur maître s'arrête pour des raisons x (dump ou lock …**1053**. « Query partially completed on the master …), la réplication s'arrête également sur le serveur esclave et ne reprends pas.

```
#replication slave
server-id=2
replicate_do_db=centrale
relay-log=/var/lib/mysql/slave-relay-bin.log

#replication master
server-id=1
binlog_do_db=centrale
log-bin=/var/lib/mysql/mysql-master-bin.log
auto_increment_increment=10
auto_increment_offset=1
```



Sur le serveur A :
auto_increment_increment = 10
auto_increment_offset = 1
Exemple : 11 – 21 - 31

Sur le serveur B :
auto_increment_increment = 10
auto_increment_offset = 2
Exemple 12 – 22 - 32

PRINCIPE DE LA RÉPLICATION

Le travail de la réplication est avant tout fait par le serveur esclave qui se connecte sur le serveur avec un utilisateur qui a le privilège **replication slave** uniquement.
Cet utilisateur devra donc être créé sur le serveur maître…

Create user 'replicant'@'%' identified by 'replicator';

Grant replication slave on *.* to 'replicant'@'%' ;

% est très important car le user se connectera en replicant @192.x.x.x

Les étapes de la réplication :

- Une requête d'écriture est joué sur le maître
- Elle est stockée dans le journal binaire du maître (**log_bin = /var/log/mysql/mysql_master_bin.log**)

- **Io_thread** la récupère et la copie sur le serveur esclave dans le journal relais (relay-log = /var/log/mysql/slave-relay.log)
- **sql_thread** l'éxécute sur le serveur slave …

- Sur la machine esclave, 2 fichiers contiennent les infos de replication

- **Ces deux fichiers se créent automatiquement lors du start slave !!**
- **master.info** : contient les infos du i/o thread (nom du journal binaire du maître, nom du user de réplication)
- **relay-log.info** : infos relatives au **sql_thread** (nom et position dans le relay-log, nb d'octets lu du journal binaire du maître)

```
[root@]$ pwd
/var/lib/mysql1
[root@]$ cat relay-log.info
/var/lib/mysql/slaveA-relay.000003
636
mysql_masterB_bin.000001
483
[root@]$ cat master.info
15
mysql_masterB_bin.000001
483
127.0.0.1
replicant
replicator
3308
60
0
```

Conf.

```
#replication slave
#server-id=2
#replicate_do_db=centrale
#relay-log=/var/lib/mysql/slave-relay-bin.log

#replication master
server-id=1
binlog_do_db=centrale
log-bin=/var/lib/mysql/mysql-master-bin.log
```

Dans le répertoire du slave, les fichiers ci-dessous se créent automatiquement lors du change master to master … et start slave

- **Master.info**
- **Relay-log.info**

REPLICATION CROISEE EN MULTI INSTANCES

```
[mysqld_multi]
mysqld     = /usr/bin/mysqld_safe
mysqladmin = /usr/bin/mysqladmin
user       = root
password   = debian

#------------------- Serveur A -----------------
[mysqld1]
socket     = /tmp/mysql.sock1
port       = 3307
pid-file   = /var/lib/mysql1/hostname.pid1
datadir    = /var/lib/mysql1
language   = /usr/share/mysql/french
user       = mysql

server-id=1
log-bin = /var/log/mysql/mysql_masterA_bin.log
binlog_do_db=centrale
replicate-do-db=centrale

master-host = 127.0.0.1
master-port = 3308
master-user = replicant
master-password = replicator

relay-log = /var/lib/mysql/slaveA-relay.log
relay-log-index = /var/lib/mysql/slaveA-relay-log.index

auto_increment_increment      = 10
auto_increment_offset         = 1
```

SLAVE

```
#-------------Serveur  B ---------------------
[mysqld2]
socket     = /tmp/mysql.sock2
port       = 3308
pid-file   = /var/lib/mysql2/hostname.pid2
datadir    = /var/lib/mysql2
language   = /usr/share/mysql/french
user       = mysql

server-id=2
log-bin = /var/log/mysql/mysql_masterB_bin.log
binlog_do_db=centrale
replicate-do-db=centrale
master-host = 127.0.0.1
master-port = 3307
master-user = replicant
master-password = replicator
master-connect-retry=60

relay-log = /var/lib/mysql/slaveB-relay.log
relay-log-index = /var/lib/mysql/slaveB-relay-log.index
auto_increment_increment      = 10
auto_increment_offset         = 2
```

Ctrl et commandes

```
1  stop slave;# MySQL a retourné un résultat vide (aucune ligne).
2  # MySQL a retourné un résultat vide (aucune ligne).
3  # MySQL a retourné un résultat vide (aucune ligne).
4
5
6  CHANGE MASTER TO MASTER_HOST='192.168.43.12',
7  MASTER_PORT=3307,
8  MASTER_USER='replication',
9  MASTER_PASSWORD='replicator';# MySQL a retourné un résultat vide (aucune ligne).
10
11
12 start slave;# MySQL a retourné un résultat vide (aucune ligne).
13
```

START SLAVE;

STOP SLAVE;

RESET SLAVE;

START SLAVE SQL_THREAD;

START SLAVE IO_THREAD;

SHOW SLAVE STATUS;