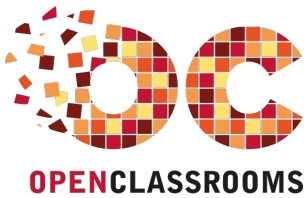
# Les moteurs de stockage de MySQL

**Par Grabeuh**




www.openclassrooms.com


*Dernière mise à jour le 6/06/2011*

# Sommaire

# Les moteurs de stockage de MySQL

Par  Grabeuh

Mise à jour : 06/06/2011
Difficulté : Intermédiaire Durée d'étude : 2 heures



MySQL est souvent décrié par de nombreux développeurs comme étant lent, ne gérant pas les relations ou les transactions, et est souvent considéré à tort comme un mauvais outil.
Mais comme tout outil, il faut savoir le manipuler pour en tirer réellement profit.

Nous allons découvrir ici quelques facettes de MySQL assez peu connues des débutants et qui peuvent dans certains cas nous changer la vie.

Ce tutoriel ne parle pas de la version Cluster de MySQL en raison de sa complexité et n'abordera donc pas le moteur de stockage NDBCluster qui lui est associé.
La documentation MySQL pourra vous renseigner largement sur ce moteur

Sommaire du tutoriel :



## MySQL et ses drôles de moteurs

On appelle moteur de stockage l'ensemble des algorithmes utilisés par un SGBDR pour stocker les informations et y accéder au moyen d'une requête SQL.

La plupart des SGBDR proposent un moteur unique, créé pour être le plus efficace possible dans tous les cas. MySQL et ses forks (dont les plus connus sont MariaDB et Drizzle) se démarquent de leurs homologues en proposant à l'administrateur de la base de choisir pour chaque table de sa base quel moteur il désire utiliser.
On se retrouve ainsi avec des bases où plusieurs moteurs peuvent coexister. C'est un choix de conception qui a ses avantages et inconvénients.
Le développeur est libre de choisir la manière dont ses données sont stockées, mais en revanche, s'il ne fait pas attention à ce qu'il fait, on peut très vite perdre pied sur de grosses bases de données et ne plus savoir au final quelle table utilise quel moteur.

Actuellement, MySQL dispose de nombreux moteurs, avec chacun une utilité particulière et des cas spécifiques d'utilisation.
Voici les principaux :

- MyISAM
- InnoDB
- MEMORY (anciennement HEAP)
- MERGE
- BLACKHOLE
- BerkeleyDB ou BDB

- ARCHIVE
- CSV
- FEDERATED

Trois autres moteurs existent mais sont soit inutilisables, soit très complexes à appréhender et sortent du cadre de ce tutoriel.

- Il s'agit pour le premier de **Isam**, moteur historique de MySQL 1.0 , et qui a été remplacé par MyISAM à partir de la version 3.23.0.
  Depuis MySQL 5.0, Isam n'est même plus inclus dans le code source de MySQL.
- Le second moteur est **NDBCluster**, qui est le type spécifique de la version clusterisée de MySQL. Comme précisé dans l'introduction, pour ne pas trop compliquer ce tutoriel je n'aborderai pas la MySQL Cluster et je vous renvois donc à la documentation de la version Cluster de MySQL pour de plus amples informations à ce sujet.
- Enfin, le moteur inutilisable est **EXAMPLE**, dont l'unique but est de permettre aux développeurs d'avoir un exemple de structure de code pour créer leur propre moteur de stockage.

De plus, par la nature modulaire des moteurs de stockages dans MySQL, des moteurs non officiels existent et peuvent être ajoutés à ceux présents de base comme s'il s'agissait de plug-ins.
C'est le cas par exemple de Percona-XtraDB. Ce moteur est l'évolution non officielle d'InnoDB et est utilisé comme son remplaçant dans MariaDB.
On citera aussi PBXT, OQGRAPH ou Aria (moteur par défaut de MariaDB).

## Le mot clé ENGINE

Non, je ne parle pas ici d'un gros mal de gorge mais du mot anglais signifiant "moteur". 

Voyons déjà quels sont les moteurs que MySQL met à notre disposition, ceci au moyen de la commande `SHOW ENGINES;`.
Cette commande nous affiche la liste complète des moteurs disponibles, une colonne Support nous signale si le moteur est activé dans MySQL (YES/NO), une colonne Comment nous renseigne sur les spécificités du moteur. Les versions les plus récentes de MySQL précisent aussi une colonne Transactions (YES/NO) si le moteur permet de gérer les transactions.

**Code : Console**

```
mysql> SHOW ENGINES;
+------------+---------+----------------------------------------------------------------
| Engine     | Support | Comment
+------------+---------+----------------------------------------------------------------
| InnoDB     | YES     | Supports transactions, row-level locking, and foreign keys
| MRG_MYISAM | YES     | Collection of identical MyISAM tables
| BLACKHOLE  | YES     | /dev/null storage engine (anything you write to it disappe
| CSV        | YES     | CSV storage engine
| MEMORY     | YES     | Hash based, stored in memory, useful for temporary tables
| FEDERATED  | NO      | Federated MySQL storage engine
| ARCHIVE    | YES     | Archive storage engine
| MyISAM     | DEFAULT | Default engine as of MySQL 3.23 with great performance
+------------+---------+----------------------------------------------------------------
8 rows in set (0.00 sec)
```

On voir dans l'exemple ci-dessus que FEDERATED est présent mais pas actif, qu'InnoDB est le seul moteur présent à gérer les transactions et que MyISAM est le moteur défini par défaut.
Maintenant que nous connaissons les moteurs disponibles sur notre système, nous allons pouvoir utiliser ceux qui nous interessent.

Les moteurs de stockage se spécifient à l'aide du mot clé `ENGINE=xxxxx` soit au moment de créer la table, soit à l'aide d'un `ALTER TABLE` :

**Code : SQL**

```
/* A la création de la table */
CREATE TABLE maTable(
...
)ENGINE=MonMoteurDeStockage;

/* En modifiant une table déjà créée */
```

```
ALTER TABLE maTable ENGINE=UnAutreMoteur;
```

Pour des raisons de compatibilité avec les anciennes versions de MySQL, on rencontre parfois le mot clé **TYPE** à la place de ENGINE.

Parfois, certaines moteurs utilisent d'autres mots clés pour spécifier des paramètres supplémentaires, comme **COMMENT**. Ces cas seront signalés plus tard dans le descriptif de chaque moteur.

Et faut le faire pour chaque table ou on peut dire à MySQL qu'on veut toujours utiliser un moteur en particulier par défaut ?

Il est possible de définir un moteur par défaut pour les nouvelles tables en le spécifiant dans le fichier de configuration de MySQL. Soit définitivement, soit pour la session active seulement.
Pour la session active seulement, il faudra utiliser

**Code : SQL**

```
SET storage_engine=NomDuMoteur;
```

Pour le faire définitivement, cela se fait au moyen de la directive suivante du fichier de configuration :

**Code : my.conf**

```
[mysqld]
default-storage-engine = NomDuMoteur
```

- Pour les versions inférieures à MySQL 5.5 le moteur par défaut est MyISAM.
- Pour les versions supérieures ou égales à 5.5, le moteur par défaut est InnoDB.

## Deux grandes familles de moteurs

Comme nous l'a montré la commande **SHOW** ENGINES, certains moteurs gèrent les transactions là où d'autres ne le font pas.

Les moteurs transactionnels sont plus surs que les moteurs non transactionnels, car ils assurent qu'une opération s'est exécutée du début à la fin sans être interrompue, et permettent d'annuler l'opération entière au cas où un incident serait survenu.

Les moteurs transactionnels offrent les sécurités suivantes :

- Si un problème matériel ou électrique survient pendant une opération et que celle-ci ne peut se terminer, les anciennes données sont récupérables et ne sont pas corrompues avec des fragments de nouvelles données.
- Vous pouvez grouper les instructions exécutées par MySQL. Il est plus simple pour lui d'effectuer plusieurs opérations identiques à la suite que d'effectuer de nombreuses opérations différentes.
- Si une mise à jour échoue, les changements sont annulés
- Les moteurs transactionnels obtiennent de meilleures performances pour les accès concurrents en lecture

Les moteurs non transactionnels offrent en contrepartie de meilleures performances, car ils ne sont pas soumis à des vérifications nombreuses

- Plus rapides
- Moins de place utilisée sur le disque
- Moins de mémoire consommée

Je vous renvois au tutoriel de Tortue Facile sur les transactions si vous voulez plus d'informations.

Même s'il est possible de le faire, il est **très fortement déconseillé** de mélanger des tables à moteurs transactionnels et des tables à moteurs non transactionnels au sein d'une même transaction.
Les tables non transactionnelles ne vont pas le devenir par magie. Si l'opération échoue, seules les tables transactionnelles pourront revenir à un état précédent, les tables non transactionnelles seront corrompues avec des

données incohérentes sans possibilité de retour en arrière.

# MyISAM

## *Présentation de MyISAM*

MyISAM est disponible dans toutes les versions de MySQL à partir de la 3.23.0.
Pour les versions antérieures à la 5.5, c'est le moteur par défaut de MySQL.

Il s'agit d'un moteur **non transactionnel** assez rapide en écriture et très rapide en lecture. Ceci vient en grande partie du fait qu'**il ne gère pas les relations ni les transactions** et évite donc des contrôles gourmands en ressources mais perd en sûreté ce qu'il gagne en vitesse.
**Il gère l'indexation des contenus**, accélérant les recherches, et il est le seul moteur de MySQL permettant de créer des index `FULLTEXT` sur les champs de type `TEXT`, rendant les recherches beaucoup plus efficaces qu'avec `LIKE %`.
Ceci en fait le moteur de prédilection pour les fonctions de recherche avancées.

De plus, MyISAM garde en cache des métadonnées sur la table et ses index, comme le nombre de lignes, la taille perdue à cause de la fragmentation, etc.
Ainsi, une requete `SELECT COUNT(*) FROM maTable;` sera extrêmement rapide avec MyISAM car le résultat est déjà obtenu à l'avance.

Ceci ne fonctionne que sur les `COUNT()` opérant sur la table entière (pas de WHERE, LIMIT ou autres restrictions)

Il est conseillé pour les tables MyISAM n'ayant pas besoin de recherches FULLTEXT d'utiliser des champs de taille fixe (utilisation de `CHAR` au lieu de `VARCHAR`, pas de `BLOB` ou de `TEXT`) afin d'améliorer la vitesse de lecture.
Ce gain, pouvant aller jusqu'à 30% de gain de performances dans des cas particuliers (attendez vous plutôt à un gain de 15% en moyenne), est contrebalancé par une taille plus importante occupée sur le disque dur.
De plus, il n'est sensible que sur des tables contenant de TRES NOMBREUX enregistrements (plusieurs millions) ou comportant de nombreuses colonnes.

Si une table comporte des champs CHAR et des champs `VARCHAR`, les champs `CHAR` seront automatiquement convertis en `VARCHAR` si leur taille est supérieure à `CHAR(3)`.
Si une table contient des `VARCHAR` d'une taille inférieure à `VARCHAR(3)`, ils seront automatiquement convertis en `CHAR`. La première règle de conversion surpasse celle-ci, si au moins un des champs `VARCHAR` de la table est supérieur à `VARCHAR(3)` les autres champs `VARCHAR` ne sont pas convertis.
Ceci vient du fait que les champs `VARCHAR` comportent des informations supplémentaires pour déterminer la taille du champ, qui sont stockées sur 2 octets. Dans le cas où un `VARCHAR` est inférieur à 3 octets, le `CHAR` est plus économique car il ne comporte pas ces informations.

La table possède un **verrouillage en lecture/écriture au niveau de la table entière**, ce qui fait que lorsqu'un utilisateur est en train d'écrire dans la table, les personnes essayant d'y accéder en lecture doivent attendre que l'écriture soit finie, et ce même si elles veulent lire une ligne autre que celle qui est en train d'être écrite.

## *Exemple d'utilisation*

Voici l'exemple d'une base de données où nous désirons enregistrer des informations sur des personne qui pourront se connecter à notre application.
Nous créons donc une table personne qui contiendra les informations sur cette personne.

**Code : SQL**

```
CREATE TABLE Personne(
Personne_id BIGINT NOT NULL PRIMARY KEY auto_increment,
Personne_username CHAR(100) UNIQUE NOT NULL,
Personne_password CHAR(40) NOT NULL,
Personne_nom CHAR(100) NOT NULL,
Personne_prenom CHAR(100) NOT NULL,
Personne_adresse CHAR(200) NOT NULL,
Personne_codepostal CHAR(5) NOT NULL,
Personne_email CHAR(200) NOT NULL,
Personne_ville CHAR(100) NOT NULL,
```

```
INDEX('Personne_nom', 'Personne_prenom')
)ENGINE=MyISAM;
```

Ici, un exemple sur les index FULLTEXT

**Code : SQL**

```
/* on crée une table contenant des news */
CREATE TABLE news(
news_id bigint not null primary key auto_increment,
news_titre varchar(200),
news_texte TEXT,
FULLTEXT(news_titre, news_text)
)ENGINE=MyISAM;

/* pour effectuer une recherche sur un mot on utilse la commande
suivante */
SELECT * FROM news WHERE MATCH (news_titre, news_texte) AGAINST
('database');
/* ceci nous renverra tous les résultats où le mot 'database'
apparait au moins une
fois soit dans le titre, soit dans le corps du texte. */
```

Des modificateurs permettent d'effectuer des recherches plus complexes par associations de termes. Voir documentation MySQL sur la recherche Fulltext pour plus d'informations

### *Avantages et inconvénients*

Avantages :

- Très rapide en lecture
- Extrêmement rapide pour les opérations COUNT() sur une table entière
- Les index `FULLTEXT` pour la recherche sur des textes

Inconvénients :

- Pas de gestion des relations
- Pas de gestion des transactions
- Bloque une table entière lors d'opérations d'insertions, suppressions ou mise à jour des données

## InnoDB

### *Présentation d'InnoDB*

InnoDB est un moteur assez performant **faisant partie de la famille des moteurs transactionnels**.
**Il ne gère pas les index en FULLTEXT** contrairement à MyISAM.

**InnoDB est un moteur relationnel.** Il s'assure que les relations entre les données de plusieurs tables sont cohérentes et que si l'on modifie certaines données, que ces changements soient répercutés aux tables liées.

**InnoDB gère le verrouillage des données au niveau de la ligne**, contrairement à MyISAM qui les gère au niveau de la table. Ainsi, si vous écrivez des données sur la première ligne de la table, seule celle-ci sera bloquée et toutes les autres seront disponibles en écriture et lecture.

InnoDB est plus lent que MyISAM, de par sa nature transactionnelle et relationnelle, même si les récentes modifications qu'il a subi pour le passage à la version 5.5 l'ont rendu beaucoup plus performant qu'avant, et ses performances sont désormais très proches de celles de MyISAM. Cependant, la sécurité qu'il apporte quant à la validité des données dans la base en fait un moteur de choix lorsqu'il s'agit de stocker des données relationnelles ou nécessitant des transactions.

### *Exemple d'utilisation*

Reprenons l'exemple que nous avions précédemment : nous désirons conserver des informations sur des personnes.
Les données contenues dans le champ Personne_ville et Personne_codepostal sont longues (105 caractères à chaque fois), et la même information peut revenir plusieurs fois (plusieurs personnes peuvent habiter à Dunkerque, par exemple).
Nous allons extraire ces données dans une table Ville qui va contenir la ville et le code postal associé, et nous laisserons une référence vers cette table dans notre table personne.
Nous allons donc utiliser le moteur InnoDB ici.

**Code : SQL**

```
CREATE TABLE Personne(
Personne_id BIGINT NOT NULL PRIMARY KEY auto_increment,
Personne_username CHAR(100) UNIQUE NOT NULL,
Personne_password CHAR(40) NOT NULL,
Personne_nom CHAR(100) NOT NULL,
Personne_prenom CHAR(100) NOT NULL,
Personne_adresse CHAR(200) NOT NULL,
Personne_email CHAR(200) NOT NULL,
Personne_Ville_id BIGINT NOT NULL REFERENCES Ville (Ville_id) ON
UPDATE CASCADE ON DELETE CASCADE,
INDEX('Personne_nom', 'Personne_prenom')
)ENGINE=InnoDB;

CREATE TABLE Ville(
Ville_id BIGINT NOT NULL PRIMARY KEY auto_increment,
Ville_codepostal CHAR(5) NOT NULL,
Ville_ville CHAR(100) NOT NULL,
INDEX('Ville_codepostal', 'Ville_ville')
)ENGINE=InnoDB;
```

### *Avantages et inconvénients*

Avantages :

- Gestion des relations
- Gestion des transactions
- Verrouillage à la ligne et non à la table

Inconvénients :

- Plus lent que MyISAM
- Occupe plus de place sur le disque dur
- Occupe plus de place en mémoire vive

**Secret** (cliquez pour afficher)

Pour la petite histoire, Oracle Corporation voyait d'un mauvais oeil l'avancée de MySQL sur ses parts de marché et avait racheté l'entreprise qui développait InnoDB pour MySQL. Oracle a continué à fournir le moteur de stockage afin de paraitre gentil et d'aider la communauté de MySQL, mais faisait tout pour ralentir son évolution, et donc limitait la concurrence envers son propre SGBD, Oracle Database.

Depuis maintenant plus d'un an, Oracle a racheté Sun, qui détenait MySQL et par là même éliminé un sérieux concurrent. Oracle donc repris le développement d'InnoDB, car il destine les deux SGBD à des secteurs différents : MySQL pour le web et Oracle Database pour les applications lourdes en entreprise.
Pratiquement du jour au lendemain, la vitesse d'exécution des requêtes a été augmentée sur de 350% sur Linux et de 1500% sur Windows. Comme quoi, quand on veut, on peut 

## MEMORY (anciennement HEAP)

### *Présentation de MEMORY*

MEMORY (HEAP est synonyme mais est déprécié, il est conseillé d'utiliser seulement MEMORY) est un moteur de stockage permettant de créer des tables directement dans la mémoire vive, sans passer par le disque dur pour stocker les données. Ceci en fait le moteur de stockage le plus rapide que propose MySQL, mais aussi le plus dangereux.

Il est à réserver seulement au stockage de données temporaires car le moindre plantage, la moindre panne ou coupure de courant fera disparaitre définitivement les données stockées avec ce moteur. Il ne restera que la table vide à la remise en marche de MySQL.

**Il ne gère pas les champs TEXT ni BLOB**. Ceux-ci prennent trop de place et donc vont contre la philosophie de ce moteur : la rapidité de traitement sur des données légères et temporaires.

**Ni les transactions, ni les relations ne sont gérées par ce moteur. Comme il ne gère pas les champs TEXT, il ne gère pas non plus les index FULLTEXT**

MEMORY est parfait pour stocker des données purement temporaires qui ont besoin d'être traitées rapidement et surtout dont la perte n'est pas significative.
L'absence de gestion de relations et de transactions n'est donc pas vraiment un problème dans ce cas. La probabilité qu'une relation entre deux enregistrements change entre le moment où la donnée temporaire est créée et le moment où elle a fini d'être traitée est très faible.
De même, aucun intérêt à gérer des transactions censées garantir que les données sont fiables dans le temps si ces mêmes données ne sont ni vitales ni censées durer dans le temps.

### *Exemple d'utilisation*

Nous voulons ajouter dans notre application une validation par email des personnes qui s'inscrivent. Nous générerons un token de validation unique et nous stockerons la personne dans une table MEMORY.
Quand la personne recevra son mail et cliquera sur le lien proposé, nous vérifierons que le token correspond bien à celui stocké dans cette base temporaire et nous déplacerons la personne vers la table réelle puis nous le supprimerons de la table temporaire.

Aucune donnée vitale à notre application n'est mise en danger, une personne non inscrite peut toujours répéter son inscription en cas de soucis, et la donnée est rapidement supprimée de la base et ne pollue donc pas la mémoire vive trop longtemps.

**Code : SQL**

```
CREATE TABLE Personne(
Personne_id BIGINT NOT NULL PRIMARY KEY auto_increment,
Personne_username CHAR(100) UNIQUE NOT NULL,
Personne_password CHAR(40) NOT NULL,
Personne_nom CHAR(100) NOT NULL,
Personne_prenom CHAR(100) NOT NULL,
Personne_adresse CHAR(200) NOT NULL,
Personne_Ville_id BIGINT NOT NULL,
Personne_Email CHAR(200) NOT NULL,
Personne_tokenvalidation CHAR(40) NOT NULL,
)ENGINE=MEMORY;
```

### *Avantages et inconvénients*

Avantages :

- Le moteur le plus rapide
- Ne consomme pas de place sur le disque dur

Inconvénients :

- Les données sont volatiles : un arret du serveur et elles disparaissent
- Pas de champs BLOB ou TEXT

## MERGE (anciemment MRG_MyISAM)

### *Présentation de MERGE*

MRG_MyISAM est un alias déprécié. Il est conseillé désormais d'utiliser seulement MERGE.
Il s'agit d'un moteur regroupant plusieurs tables MyISAM de manière transparente. Les tables fusionnées existent encore indépendamment, MERGE se contente de fournir une interface unique pour accéder en lecture à toutes les tables simultanément, et en écriture selon des règles que l'on aura fixé.

Les tables MyISAM fusionnées avec MERGE peuvent provenir de plusieurs bases de données, tant qu'elles sont sur le même serveur physique.
MERGE gère les index de la même manière que MyISAM, sauf pour les index FULLTEXT qu'il ne prend pas en compte.

Les tables fusionnées doivent respecter plusieurs critères pour pouvoir être fusionnées avec MERGE :

- Être sur le même serveur
- Mêmes noms de champs
- Mêmes types pour ces champs
- Mêmes index (sauf FULLTEXT qui sera ignoré)
- Même ordre de déclaration des index

Chaque table individuelle reste disponible, ce qui permet d'effectuer des opérations dessus, qui se répercuteront sur la table fusionnée.
On peut alors facilement faire des recherches sur une base spécifique.

MERGE fait partie des moteurs utilisant 2 paramètres supplémentaires dans sa déclaration en plus de `ENGINE=MERGE` :

- **`UNION`**`(t1, t2, t3, ...)` : on donnera la liste des tables à fusionnner.
- `INSERT_METHOD=VALEUR` : Définit où seront insérées les nouvelles valeurs. Les valeurs possibles sont **`FIRST`** et **`LAST`**. **`FIRST`** insérera les données dans la première table de la liste alors que **`LAST`** les insérera dans la dernière table de la liste

Pour rajouter une nouvelle table à la fusion, il existe trois solutions :

- Supprimer la table à l'aide d'un **`DROP`** puis la recréer avec la nouvelle table en plus
- utiliser la syntaxe **`ALTER TABLE`** `nom_de_la_table` **`UNION`**`=(table1 , table2 , ... , nouvelle_table)`
- Modifier à la main le fichier .MRG créé dans le dossier de données de MySQL puis faire un `FLUSH TABLES` dans le SGBD pour le forcer à relire les définitions (méthode à réserver aux experts)

### *Exemple d'utilisation*

Nous avons créé un 2$^{eme}$ site en plus du premier où nous utilisions notre table Personne.
Cependant, nous nous rendons compte que nos utilisateurs du 1$^{er}$ site sont potentiellement intéressés par le second que nous venons d'ouvrir, aussi nous décidons de rendre leur donner accès au nouveau site sans avoir besoin de s'inscrire.
Les personnes s'inscrivant sur le 2$^{eme}$ site ne sont par contre pas spécialement intéressées par le 1$^{er}$, nous voulons donc qu'elles se créent elles-mêmes un compte sur le 1$^{er}$ site.
Notre système doit donc gérer les lectures sur les 2 tables, et les écritures seulement sur la 2$^{eme}$ table.
Nous pouvons faire ceci très facilement avec une table MERGE avec `INSERT_METHOD=LAST`.

**Code : SQL**

```
USE baseSite1;

/* Nous créons la table du site 1 */
CREATE TABLE PersonneSite1(
Personne_id BIGINT NOT NULL PRIMARY KEY auto_increment,
Personne_username CHAR(100) UNIQUE NOT NULL,
Personne_password CHAR(40) NOT NULL,
Personne_nom CHAR(100) NOT NULL,
Personne_prenom CHAR(100) NOT NULL,
```

```
Personne_adresse CHAR(200) NOT NULL,
Personne_codepostal CHAR(5) NOT NULL,
Personne_email CHAR(200) NOT NULL,
Personne_ville CHAR(100) NOT NULL,
INDEX('Personne_nom', 'Personne_prenom')
)ENGINE=MyISAM;

/* Nous créons la table du site 2 dans une base distincte mais sur
le même serveur
Elle contient les mêmes champs que la table du site 1*/
USE baseSite2;

CREATE TABLE PersonneSite2(
Personne_id BIGINT NOT NULL PRIMARY KEY auto_increment,
Personne_username CHAR(100) UNIQUE NOT NULL,
Personne_password CHAR(40) NOT NULL,
Personne_nom CHAR(100) NOT NULL,
Personne_prenom CHAR(100) NOT NULL,
Personne_adresse CHAR(200) NOT NULL,
Personne_codepostal CHAR(5) NOT NULL,
Personne_email CHAR(200) NOT NULL,
Personne_ville CHAR(100) NOT NULL,
INDEX('Personne_nom', 'Personne_prenom')
)ENGINE=MyISAM;

/* Enfin on crée la table fusionnée qui comporte strictement les
mêmes champs et indexes.
On précise que l'union se fait sur les 2 tables et qu'on veut
insérer dans la dernière table
la liste
Enfin, elle est créée dans la baseSite2 donc ne sera pas accessible
dans baseSite1*/
CREATE TABLE Personne(
Personne_id BIGINT NOT NULL PRIMARY KEY auto_increment,
Personne_username CHAR(100) UNIQUE NOT NULL,
Personne_password CHAR(40) NOT NULL,
Personne_nom CHAR(100) NOT NULL,
Personne_prenom CHAR(100) NOT NULL,
Personne_adresse CHAR(200) NOT NULL,
Personne_codepostal CHAR(5) NOT NULL,
Personne_email CHAR(200) NOT NULL,
Personne_ville CHAR(100) NOT NULL,
INDEX('Personne_nom', 'Personne_prenom')
)ENGINE=MERGE UNION(baseSite1.PersonneSite1,
baseSite2.PersonneSite2) INSERT_METHOD=LAST;
```

### *Avantages et inconvénients*

Avantages :

- Permet de gérer facilement des historiques : on place les données de chaque mois dans une nouvelle table qu'on fusionne avec les autres tables à l'aide de MERGE. Les données les plus récentes sont toujours envoyées dans la table la plus récente.
- Permet de regrouper les données de plusieurs sites en une seule table comme dans l'exemple
- MySQL fournit des fonctions de réparations de tables pour les fichiers endommagés. Il est possible de fusionner les tables se ressemblant pour les réparer plus efficacement car l'opération sera faite en une fois et les étapes communes du processus seront faites une seule fois.
- On peut créer des alias de tables en fusionnant une seule table dans une table MERGE, utile pour assurer une rétrocompatibilité tout en faisant évoluer son application

Inconvénients :

- MERGE est plus gourmand en ressources système que MyISAM. Pour 1 table MERGE contenant 10 tables MyISAM avec chacune au moins un fichier d'index, vous vous retrouvez avec 21 descripteurs de fichier (1 table + 10 tables + 10 index). Multipliez ceci par le nombre d'utilisateurs qui utilisent la base et vous risquez de saturer assez vite le système.
- Les recherches sur les tables contenant des index sont plus lentes que sur des tables individuelles, car doit MERGE

consulter les index de toutes les tables qu'il fusionne.
- Les tables fusionnées doivent être identiques au niveau de leur structure et de l'ordre de déclaration des index

## Autres moteurs

Les moteurs suivants sont rarement utilisés soit parce qu'ils ont été remplacés par de nouveaux moteurs ayant les mêmes fonctions mais plus performants, soit parce qu'ils sont limités à des cas très particuliers. Il est cependant bon de les connaitre au cas où vous rencontreriez une situation où vous en avez besoin.

### *BLACKHOLE*

Ou en français : trou noir.
Ce moteur porte bien son nom, ce qu'on y stocke n'en ressort jamais, et pour cause, il est l'équivalent du /dev/null d'Unix/Linux.

Toute données envoyées dans une table BLACKHOLE est immédiatement détruite, mais l'action est cependant consignée dans les logs de MySQL.

Ce moteur de table est utile pour simuler des copies de tables ou vérifier la syntaxe d'un système de sauvegarde.
Il permet entre autres de recherche facilement des goulots d'étranglement dans les requêtes SQL d'une application dans écrire réellement sur le disque, ou lorsqu'on désire tester les performances du système de logs de MySQL.
On l'utilise aussi parfois afin de diminuer le trafic réseau lorsqu'on exécute une réplication de serveurs à des fins de sauvegarde (comme la réplication se base sur les logs de requêtes, le serveur copie croit que le serveur original est actif et se synchronise).
Je vous renvois pour ça à la documentation de MySQL pour de plus amples informations.

### *BerkeleyDB (BDB)*

BDB est un moteur assez peu utilisé, car assez lent, et dont les fonctionnalités intéressantes sont équivalentes à celles proposées par InnoDB.
**Son principal défaut est de ne pas être inclus dans les versions standard de MySQL et de ne pas être non plus compatible avec certains systèmes d'exploitation, dont MacOS X et certaines versions 64bits du noyau Linux.**
Il est par contre fourni en standard avec la distribution de MySQL de tous les Unix de la famille BSD (OpenBSD, NetBSD et FreeBSD).
Il faut soit compiler MySQL avec ce moteur, soit prendre un binaire MySQL-max déjà compilé avec cette option.
La version Windows de MySQL comprennait BDB en standard jusqu'à MySQL 5.5. Il a depuis été retiré de la distribution.

Ce moteur était surtout utile à l'époque des premières versions de MySQL, lorsqu'InnoDB n'existait pas. Il fonctionne de la même manière et propose les mêmes outils majeurs :

- gestion des transactions
- gestion des relations
- verrous de ligne au lieu de verrous de tables

Même s'il est toujours maintenu, sa lenteur et ses problèmes de compatibilité l'éliminent pratiquement d'office. Cependant, sur des vieux serveurs MySQL, on peut le rencontrer.

### *ARCHIVE*

ARCHIVE est un moteur spécialisé dans le stockage de grosses quantités de données de manière très économique : les données sont compressées à leur insertion et aucun index n'est généré, ce qui améliore la rapidité en écriture.
**Il ne gère ni les transactions, ni les relations ni les index**, et ne permet de faire que des requete `SELECT` et `INSERT`. Les ordres de suppression ou de modifications seront refusés.

On peut ainsi conserver d'énormes quantités de données sans craindre qu'elles soient supprimées ou modifiées.

### *CSV*

Il s'agit du format de données basique utilisé habituellement par les tableurs. Les valeurs sont stockées dans un fichier texte, séparées par des virgules, les lignes sont séparées par des sauts de ligne. **Le moteur ne gère ni relations, ni transactions, ni index**.

Il permet une grande interopérabilité entre des systèmes externes à MySQL, en travaillant sur un format de données pratiquement universel, reconnu par tous les tableurs et de nombreux logiciels qui importent et exportent des données dans ce

format.

Il est en outre extrêmement pratique lorsqu'on désire exporter une table au format CSV compatible avec Excel/OOo-Calc, il suffit de faire une copie de la table avec ce moteur pour la nouvelle table. Mais ceci nécessite d'avoir accès aux dossiers où sont contenus les fichiers de stockage de MySQL.

### *FEDERATED*

FEDERATED est un moteur assez complexe à appréhender. Il crée une définition de table, mais rien n'est stocké directement sur le serveur.
Il s'agit d'un moteur de stockage distant : les données sont en réalité hébergées sur un autre serveur MySQL.
Dans un premier temps, il fait une requête **SELECT * FROM...** sur le serveur distant puis reconstruit une table temporaire en local, sur laquelle il exécute la requête de l'utilisateur.

Le moteur gère les commandes **SELECT**, **INSERT**, **UPDATE** et **DELETE**, ainsi que les index.

FEDERATED utilise un champ supplémentaire pour spécifier la table utilisée sur le serveur distant.
COMMENT='mysql://user@nomDuServer:port/federated/nomDeLatable';

Ce moteur est assez contraignant :

- Il n'est pas activé par défaut, il faut démarrer le serveur MySQL avec le paramètre `--federated` pour en profiter
- Pas de gestion des transactions
- La table locale n'est pas prévenue si des modifications sont faites sur la table distante, ceci peut provoquer des incompatibilités ou des incohérences
- Impossibilité de modifier la table en local, il faut forcément le faire avec le serveur distant
- Le cache de requêtes de MySQL ne prend pas ces tables en compte, ce qui ralentit encore plus le traitement des requêtes.
- Si la connexion entre les deux serveurs est impossible, la table sera inaccessible

FEDERATED s'utilise de la sorte :

**Code : SQL**

```
/*table MyISAM sur le serveur1*/
CREATE TABLE test_table (
    id      int(20) NOT NULL auto_increment,
    name    varchar(32) NOT NULL default '',
    other   int(20) NOT NULL default '0',
    PRIMARY KEY  (id),
    KEY name (name),
    KEY other_key (other)
)
ENGINE=MyISAM;

/*table FEDERATED sur le serveur2*/
CREATE TABLE federated_table (
    id      int(20) NOT NULL auto_increment,
    name    varchar(32) NOT NULL default '',
    other   int(20) NOT NULL default '0',
    PRIMARY KEY  (id),
    KEY name (name),
    KEY other_key (other)
)
ENGINE=FEDERATED
COMMENT='mysql://root@server1:9306/federated/test_table';
```

Federated sera utilisé pour des cas où l'accès à une table doit se faire entre plusieurs serveurs. Par exemple une table utilisateurs commune à plusieurs serveurs dans une entreprise.

## Essence ou Diesel ? Quel moteur choisir ?

Nous l'avons vu, les moteurs disponibles sont nombreux.
Chacun est performant dans une situation donnée mais pourrait être source de problèmes dans d'autres cas.

Dans la plupart des cas, le choix sera à faire entre MyISAM, InnoDB et MEMORY, les autres moteurs étant plus spécifiques.

Le raisonnement à tenir est le suivant :

Est-ce que mes données sont temporaires et peu importantes ?

Si les données sont temporaires et ne sont pas vitales à l'application, MEMORY sera un bon choix pour sa volatilité.

Est-ce que mes données sont en relations avec des données d'autres tables ?

Si les données sont liées à d'autres tables, on choisira InnoDB pour son respect des relations.

Est-ce que mes données doivent rester à tout prix intègres et ne pas contenir d'incohérences ?

Si la cohérence de la base de données et son intégrité sont primordiales, InnoDB sera choisi pour son respect des transactions.

Est-ce que je dois effectuer des recherches sur des textes de taille importante ?

Si des recherches sur des textes de grande taille sont à faire, MyISAM et ses index FULLTEXT seront un choix judicieux.

Est-ce que je dois faire plus d'insertions, modifications ou suppressions que de lectures dans ma table ?

MyISAM verrouille la table entière à chaque insertion, modification ou suppression. Si de nombreuses opérations de ce type sont faites, il faudra plutôt s'orienter vers InnoDB qui verrouille indépendamment chaque ligne et évite ainsi de ralentir inutilement l'application.

Est-ce que j'ai besoin d'historiser mes données ?

S'il est important de pouvoir historiser les données, MERGE sera interessant pour sa possiblité d'agrandir pratiquement à l'infini la table tout en structurant les données en sous-tables.
Les moteurs de stockage de MySQL sont des outils puissants lorsqu'ils sont utilisés de manière pertinente. MySQL donne au développeur la liberté de faire tout et n'importe quoi (la tendance naturelle de l'homme sera évidemment de faire n'importe quoi  ).
Il est ainsi important de connaitre les outils à notre disposition pour pouvoir choisir le plus adapté à la situation. On ne plante les carottes dans son jardin ni avec un cure-dents, ni avec une pelleteuse.

Il ne faut pas tomber dans le travers classique de vouloir utiliser un moteur partout parce qu'il est mieux dans 50% des cas. Dans les 50% restants, il sera inadapté.
Toujours utiliser un moteur un peu lent "parce qu'il gère mieux les relations en général" alors qu'on travaille dans une base ne possédant pas de relations serait idiot et ralentirait l'application inutilement.

Comme dit le proverbe :

**Citation**

> Lorsqu'on a un marteau à la main, tout ressemble à un clou.

C'est donc tout un travail de réflexion qui permettra de choisir le bon moteur pour la bonne table au sein d'une base. Plusieurs

peuvent coexister sans pour autant poser de problèmes.
La question n'est donc pas *"Quel est le meilleur moteur ?"* mais : "*Dans ce cas précis, quel moteur m'apportera les fonctionnalités dont j'ai besoin ?*"

### *liens utiles*

- Documentation MySQL relative aux moteurs et son article spécifique à InnoDB
- Les moteurs de stockage de MariaDB, le fork de MySQL
- Les moteurs de base de données sur Wikipedia article plus généraliste parlant aussi des autres SGBD