Les différents moteurs de stockage de MySQL

par Sony NOEL

Date de publication : 10/09/2008

Dernière mise à jour :

MySQL, contrairement aux autres SGBD, a la possibilité d'utiliser plusieurs moteurs de stockage dans une seule et même base de données. Cette faculté constitue un des points forts de **MySQL**. Cet article a pour but de vous montrer ce qui se cache derrière ces différents moteurs et comment les exploiter au mieux.

I - Présentation....4
II - Le moteur MyISAM....4
II-A - Description....4
II-B - Recherche FULL-TEXT (Texte Intégral)....5
II-B-1 - Description....5
II-B-2 - Exemples....5
II-B-3 - Conclusion sur la recherche FULL-TEXT....6
II-C - Avis....6
II-D - Conclusion....7
III - Le moteur Memory (Heap)....7
III-A - Description....7
III-B - Avis....8
III-C - Conclusion....8
IV - Le moteur Archive....8
IV-A - Description....8
IV-B - Avis....9
IV-C - Conclusion....9
V - Le moteur CSV....9
V-A - Description....9
V-B - Fonctionnalités....10
V-C - Conclusion....10
VI - Le moteur Merge....10
VI-A - Description....10
VI-B - Exemple d'utilisation....10
VI-C - Fonctionnalités....12
VI-D - Conclusion....12
VII - Le moteur Example....12
VII-A - Description....13
VII-B - Fonctionnalités....13
VII-C - Conclusion....13
VIII - Le moteur BlackHole....13
VIII-A - Description....14
VIII-B - Exemples d'application....14
VIII-B-1 - Base de référence de performances....14
VIII-B-2 - Proxy MySQL....14
VIII-B-3 - Déclencheur de cache de données....15
VIII-B-4 - Recherche du coupable pour une opération gourmande....16
VIII-C - Fonctionnalités....16
VIII-D - Conclusion....16
IX - Le moteur FEDERATED....16
IX-A - Description....16
IX-B - Exemple de mise en place d'une structure Federated....17
IX-D - Fonctionnalités....18
IX-C - Conclusion....19
X - Le moteur InnoDB....19
X-A - Description....19
X-B - Organisation interne....19
X-C - Les clés étrangères....19
X-D - Le COMMIT et ROLLBACK....21
X-E - Fonctionnalités....22
X-F - Conclusion....22
XI - Le moteur BerkeleyBD (BDB)....22
XI-A - Description....23
XI-B - Fonctionnalités....23
XII - Le moteur Falcon....23
XII-A - Description....23
XIII - Le moteur Solid DB....24
XIII-A - Description....24

XIII-B - Conclusion....24
XIV - Le moteur PBXT (Prime Base XT)....24
XV - Le moteur AWS S3 (Simple Storage Service) Amazon S3....25
XV-A - Description....25
XV-B - Conclusion....25
XVI - Le moteur NitroDB for MySQL....25
XVI-A - Description....26
XVII - Le moteur BrightHouse for MySQL....26
XVII-A - Description....26
XVIII - Le moteur Maria....26
XVIII-A - Description....26
XIX - Conclusion....26
XX - Remerciements....27

I - Présentation

De par sa souplesse, sa rapidité et sa simplicité, MySQL gagne en popularité parmi les applications du web, ainsi que sa place parmi les grands SGBD. Beaucoup de personnes pensent que MySQL ne gère pas les transactions, alors que nous disposons de plusieurs moteurs de stockage. Certains transactionnels d'autres non.
En effet, MySQL possède une API permettant de développer soi-même un moteur de stockage, sans avoir à le coder directement dans les binaires. Certaines sociétés ont développé des moteurs spécifiques aux usages que vous pouvez avoir de votre base de données. Certaines entreprises ont développé leur propre moteur de stockage, afin qu'il corresponde à ce qu'elles attendaient d'un SGBD.
Si vous le souhaitez, vous aussi vous pouvez créer votre propre moteur. Pour l'instant, nous allons vous présenter les principaux moteurs de stockage existants.

Il y a trois grandes familles de moteurs

- Non-transactionnels dits **NTST**.
- Transactionnels dits **TST**.
- Les autres.

II - Le moteur MyISAM

- **Licence :** GPL
- **Version de MySQL :** Depuis la version 3.23, MyISAM a remplacé le moteur ISAM
- **Type :** Moteur Non transactionnel.
- **Domaines d'application :**
  - Recherche FULL-TEXT (texte intégrale).
  - Tables en lecture seule.
  - Tables de Log.
- **Information :** **Documentation officielle de MyISAM**

II-A - Description

Depuis sa création, il est devenu le moteur par défaut de **MySQL**. Il a remplacé **Isam** en y ajoutant des extensions. En raison de sa souplesse, simplicité et rapidité, **MyISAM** gagne en popularité dans les applications du web.

Il est très utilisé pour le web car, comme il ne gère ni les clés étrangères, ni les transactions, il n'a pas à vérifier la validité des enregistrements. Cela permet donc un précieux gain de temps sur des tables très fréquemment ouvertes en écriture/lecture.

En effet, lorsque vous faites des suppressions sur des champs de type **VARCHAR**, **CHAR**, **BLOB** ou **TEXT**, le moteur supprime le contenu mais la place précédemment supprimée est conservée et peut être réutilisée ultérieurement.
OPTMIZE va défragmenter la table afin de gagner de la place et ainsi faciliter l'accès aux données sur cette table.

```
OPTIMIZE maTable;
```

Rien ne sert d'exécuter cette commande sur toutes les tables en permanence. Vous risquez de faire baisser les performances si beaucoup de connexions s'effectuent en même temps.
Cette commande doit être utilisée seulement sur des tables dont la taille évolue rapidement.

*D'après ses détracteurs, **MySQL** ne serait pas capable de gérer les transactions.*

*En réalité, les personnes ayant utilisé MySQL ne savaient pas qu'il était possible de changer de moteur. Ils sont tombés sur le moteur par défaut, **MyISAM**, qui ne gère pas les transactions.*

Une table MyISAM utilise trois fichiers :

- maTable.FRM : Fichier de définition de la table
- maTable.MYD : Fichier contenant les données de la table
- maTable.MYI : Fichier d'index

II-B - Recherche FULL-TEXT (Texte Intégral)

II-B-1 - Description

L'utilisation de la recherche **FULL-TEXT** est un des grands avantages de **MyISAM**.
Lorsque nous souhaitons rechercher un mot dans une table, nous pensons par défaut à l'opérateur de recherche **LIKE** et à ses différents jokers.

Exemple avec l'opérateur LIKE

```
SELECT * FROM maTable WHERE monChamp LIKE '%ma recherche%'
```

Cette requête nous retournera toutes les lignes où le champ "monChamp" contiendra "ma recherche". Elle va nous retourner les résultats dans l'ordre de la table ou dans l'ordre défini par un **ORDER BY**. **LIKE** présente deux problèmes. En mode "contient", comme nous l'avons utilisé, il ne peut s'appuyer sur un index et est donc très peu efficace. De plus, il recherche des séries de caractères : par exemple, une recherche sur %bosse% donnera nom seulement bosse, mais aussi bosser, cabosser, Carabosse, etc. A l'inverse, la recherche **FULL-TEXT** s'appuie sur un index ad hoc et recherche des mots et non des caractères.

En effet, la requête va vous retourner les enregistrements sans distinction sur la pertinence de l'information recherchée. Si vous allez dans la page d'index d'un livre, par exemple sur l'encyclopédie des bases de données et que vous cherchez le mot "**MySQL**", nous supposons que vous souhaitez tomber sur le chapitre qui en parle le plus.

MySQL, avec la recherche **FULL-TEXT**, a la possibilité d'effectuer une recherche sur une colonne ou un groupe de colonnes en retournant le résultat selon la pertinence du mot recherché. Dans le cas de notre livre, **MySQL** pourra retourner le chapitre où se trouve le plus souvent le mot "MySQL". Les résultats seront retournés, un peu à l'instar de Google et son Ranking, par ordre de pertinence.

Vous pouvez aussi faire une recherche sur une phrase. **MySQL** renverra la pertinence de chaque mot. Il va encore plus loin car il exclut les mots n'ayant pas de pertinence et redondants dans la langue. Par exemple, il ne prend pas en compte les mots de longueur inférieure ou égale à 4 caractères (des mots comme "de", "la", etc. en français).

Le nombre de lettres minimal que doit comporter un mot peut être paramétré dans le fichier de configuration.

II-B-2 - Exemples

Exemple d'utilisation de la recherche FULL-TEXT

```
CREATE TABLE `article` (
  `idArticle` INT(10) UNSIGNED NOT NULL AUTO_INCREMENT,
  `Titre` VARCHAR(250) CHARACTER SET latin1 DEFAULT NULL,
  `Article` TEXT CHARACTER SET latin1,
  PRIMARY KEY  (`idArticle`),
  FULLTEXT KEY `Titre` (`Titre`,`Article`)
) ENGINE=MyISAM AUTO_INCREMENT=0 DEFAULT CHARSET=utf8;
```

Exemple d'utilisation de la recherche FULL-TEXT

```
ENGINE = MyISAM;
```

Ajout de l'article en cours par moteur

```
INSERT INTO Article (Titre, Article) VALUES
 ('MyISAM', 'Ce moteur est une version évolué de ISAM avec des extensions en plus....');
INSERT INTO Article (Titre, Article) VALUES
 ('Memory', 'Les tables de type Memory stock les enregistrements dans la mémoire physique...');
INSERT INTO Article (Titre, Article) VALUES
 ('CSV', 'Ce type de format facilite le transport entre différentes sources ...');
```

MATCH considère que les mots présents dans au moins la moitié des lignes sont trop fréquents pour être pertinents. En conséquence, la recherche **FULL-TEXT** ne renverra aucun résultat si la table n'a pas au moins trois lignes

Dans la requête ci-dessus, le contenu de l'article est enregistré dans la base de données. Maintenant, nous pouvons procéder à une recherche avec pertinence.

```
SELECT *, MATCH (Titre, Article) AGAINST ('base de données') AS Score FROM Article ORDER BY score DESC;
-------------------------------------------------
IdArticle | Titre     | Article                                          | Score
7         | Federated | Le moteur Federated permet de déporter...        | 0,21985759722453
6         | Exemple   | Ce type de table est assez particulier...        | 0,15944281721118
8         | InnoDB    | C'est le moteur transactionnel le plus utilisé... | 0,14023887676722
1         | MyIsam    | Ce moteur est une version évoluée d'ISAM avec...  | 0,068685997386924
2         | Memory    | Les tables de type Memory stockent les...        | 0
...
```

Dans le résultat de la requête, vous remarquez que le champ score indique la pertinence estimée par **MySQL**. Avec cette information vous pouvez renvoyer à l'utilisateur un résultat proche de ce qu'il demande.

II-B-3 - Conclusion sur la recherche FULL-TEXT

La recherche **FULL-TEXT** permet d'avoir une approche utilisateur d'un vrai moteur de recherche tel que nous en trouvons sur le web.

L'exemple présenté dans cet article n'est qu'une faible partie de ce que **MyISAM** peut faire. Il y a beaucoup d'autres fonctionnalités qui peuvent être utilisées, telles que l'utilisation des expressions régulières présentes dans **MySQL** avec la recherche **FULL-TEXT**.

MySQL est capable d'effectuer une recherche par "sous-entendu" ce qui est appelé "recherche aveugle". Si à la place de "Base de données" nous plaçons "moteur", MySQL est capable de comprendre que nous parlons de **MyISAM**, **Memory**, **InnoDB**, etc. Ceci via le mot-clé **WITH QUERY EXPANSION.**

À savoir à l'heure actuelle, seul ce moteur possède cette fonctionnalité.

Pour plus d'informations sur le FULL-TEXT

II-C - Avis

Avantages

- Moteur rapide.
- Possibilité d'écrire et lire en même temps sans risque de verrouillage de table.
- Verrouillage de table manuel.
- La mise en cache des clés.

- Gain de place sur le disque.
- Gain de mémoire lors des mises à jour.
- Gestion de la recherche **FULL-TEXT**.

Inconvénients

- Pas de gestion des contraintes de clés étrangères
- Pas de gestion des transactions (pas de **COMMIT / ROLLBACK** possible).

II-D - Conclusion

MySQL AB a fait le choix de mettre ce moteur par défaut en raison de sa polyvalence et sa simplicité d'utilisation. La gestion de la recherche **FULL-TEXT** offre une grande utilité pour la création de moteurs de recherches avancés. **MyISAM** est donc le moteur le plus utilisé.

III - Le moteur Memory (Heap)

- **Licence :** GNU MySQL
- **Version de MySQL :** 3.23.25
- **Type :** Moteur non-transactionnel
- **Domaines d'application :**
  - Données volatiles
  - fort besoin en accès rapide aux données ; données temporaires
- **Information :** **Documentation officielle de Memory (HEAP)**

III-A - Description

Les tables de type Memory enregistrent les données dans la mémoire vive de la machine (RAM), d'où un gain considérable de rapidité pour accéder aux informations.

Le moteur **Memory** est extrêmement utile pour les tables très souvent sollicitées.
Attention tout de même : il serait facile de se dire que parce que nous avons beaucoup de **RAM**, nous pouvons passer toutes les tables en **Memory** ; ce n'est hélas pas si simple, car il y a aussi des inconvénients. En effet, seule la structure des tables est stockée dans des fichiers, alors que les données ne le sont qu'en mémoire. Si la machine s'arrête ou que le service de **MySQL** redémarre, vous perdez tous vos enregistrements.

Il est possible lors du démarrage du serveur avec l'option *init-file* ou dans le fichier de configuration, ou encore par ligne de commande, de remplir les tables Memory. Dès lors, vous pouvez mettre les commandes **INSERT INTO**, **SELECT** et **LOAD DATA INFILE** afin de lire des enregistrements d'une source de données persistante (fichier ou table d'un autre moteur).

Pour des raisons de sécurité du serveur, il est possible de déterminer la taille maximale allouée aux tables par la variable système *max_heap_table_size*. Vous pouvez aussi, lors de la création de la table, indiquer le nombre maximum d'enregistrements via la clause **MAX_ROWS**.
Une table **Memory** ne peut être transformée en table physique par la commande ALTER, contrairement à celles des autres moteurs.

Pour libérer de la mémoire, il suffit simplement d'exécuter les commandes **DELETE**, **TRUNCATE** ou **DROP heap_table**.

Il faut vous mettre en garde dans le cas où vous êtes dans un hébergement mutualisé.

*Beaucoup d'hébergeurs proposent ce moteur de base de données, mais il est rarement possible d'accéder au fichier de configuration de **MySQL** pour gérer la limite de la mémoire, ni le chargement des données au démarrage du serveur. Il y a une astuce qui consiste à utiliser une requête de copie.*
*Pour cela, il faut avoir une table de type **MyISAM** (ou tout autre type de table physique) de même structure que la table Memory. La requête est :*

```
INSERT INTO maTableMemory SELECT * maTableMySIAM
```

Cette astuce n'est pas optimisée ni très pratique car il faut vérifier fréquemment le contenu de la table **Memory**. Il est aussi possible de passer par un trigger qui peut vérifier l'existence de contenu de la table **Memory**.

III-B - Avis

Avantages

- Enorme gain de rapidité.
- Couramment proposé chez les hébergeurs en mutualisé mais...

Inconvénients

- Nécessité d'avoir accès au fichier de configuration de MySQL chez les hébergeurs en mutualisé.
- Perte des données en mémoire au redémarrage du serveur (même s'il existe une commande permettant de recharger les données, il faut être sûr qu'elles soient à jour dans le fichier de données.
- Impossibilité de transformer une table Memory en table physique (exemple : Memory vers MyIsam).
- Ne supporte pas les champs de type **BLOB/TEXT**, les champs **AUTO_INCREMENT** ainsi que les champs index n'étant pas en **NOT NULL**.
- Vue son utilisation, nécessite une quantité de mémoire RAM conséquente. Si la table utilise plus de mémoire que la machine en possède, l'OS va "swapper". Ce qui entraînera des performances bien plus mauvaises qu'avec l'utilisation d'une table physique.
- Possède les mêmes inconvénients que les tables **MyISAM**.

III-C - Conclusion

L'utilisation de ce moteur est très utile lorsque l'on a des données très souvent sollicitées et possédant une taille inférieure à la quantité de RAM disponible sur la machine. Un exemple d'utilisation pratique serait l'enregistrement d'objets sérialisés dans **MySQL**. Attention toutefois à ne pas dépasser la capacité de mémoire et de prévoir les redémarrages du serveur **MySQL**.

IV - Le moteur Archive

- **Licence :** GNU MySQL
- **Version de MySQL :** 4.1.3
- **Type :** Moteur non-transactionnel
- **Domaines d'application :**
  - Enregistrement de logs
- **Information :** **Documentation officielle du moteur ARCHIVE**

IV-A - Description

Les tables de types ARCHIVE sont principalement utilisées pour stocker des données brutes. Ce moteur est très adapté pour les enregistrements de logs. Il permet d'enregistrer une grande quantité de données en prenant un minimum de ressources.

Il n'y a pas de problème de collision de données, de stockage temporaire, de validation de clé et les lignes sont verrouillées. Seules deux commandes sont possibles pour ce moteur : **SELECT** et **INSERT**. La mise à jour de données **(UPDATE)** et la suppression **(DELETE)** ne sont pas possibles. En effet, il n'y a aucune raison de mettre à jour ou de détruire des lignes de logs. Si toutefois, il est nécessaire de vider la table, il faut passer par le moteur **MyISAM**.

Ce moteur ne supporte pas les index.

IV-B - Avis

Avantages

- Permet l'écriture de logs au format brut
- Ne prend pas beaucoup d'espace, vu son format d'enregistrement
- Grandes performances en écriture

Inconvénients

- Il est difficile de trouver des inconvénients si ce moteur est utilisé pour ce qu'il est censé faire, c'est-à-dire l'enregistrement de logs.

IV-C - Conclusion

Ce moteur est fait pour enregistrer les données et se faire oublier, d'où son utilité pour une table de logs. Après de nombreux tests effectués, les performances se sont avérées très acceptables.
Si la clause **LOW_PRIORITY** est ajoutée, celles-ci seront encore améliorées.

V - Le moteur CSV

- **Licence :** GNU MySQL
- **Version de MySQL :** 4.1.4
- **Type :** Moteur non-transactionnel.
- **Domaines d'application :**
  Exportation et importation de données venant d'autres sources d'informations.
- **Information :** **Documentation officielle du moteur CSV**

V-A - Description

Le moteur **CSV** enregistre les données dans un fichier texte avec une virgule en caractère séparateur.

Ce type de format facilite le transport entre différentes sources d'informations ainsi que la lisibilité par une personne. Lors de la création de la table, le serveur créé un fichier de définition de la table **maTable.frm**. Les données sont stockées dans un fichier **CSV** portant le même nom que la table. Elle supporte les opérations d'écriture basiques telle que les commandes **INSERT**, **DELETE** et **UPDATE**. Il est évident, vu le format, que ce moteur ne gère pas l'indexation.

Comme le moteur **CSV** exploite un fichier brut, il y a une dégradation des performances lors des exécutions de requêtes en lecture et écriture. Les dégradations augmenteront à mesure que la table grandit. Une astuce consisterait à utiliser une table **MyISAM** pour effectuer les requêtes de lecture et écriture. Puis, lors d'une exportation au format CSV effectuer une copie de la table au format MyISAM vers le format CSV

Il suffit de faire une copie du fichier portant le nom de la table et ayant l'extension **.CSV**, pour exporter vers une autre application.

V-B - Fonctionnalités

Avantages

- Facilement exportable vers une autre application. Pas besoin d'un langage tiers pour convertir les données au format CSV. L'importation nécessite plus de délicatesse car il faut faire correspondre les données avec les informations présentes dans le fichier de définition de la table ***.frm**.
- Données directement lisibles pour un humain.

Inconvénients

- Pertes importantes de performances sur les tables ayant beaucoup d'enregistrements.
- Ne gère pas l'indexation.
- Importation délicate.

V-C - Conclusion

Ce moteur vous sera fort utile si vous devez gérer des importations/exportations sans passer par un langage tiers.

VI - Le moteur Merge

- **Licence :** GPL
- **Version de MySQL :** 3.23.25
- **Type :** Autre
- **Domaines d'application :** Administration de base de données.
- **Information :** **Documentation officielle du moteur Merge**

VI-A - Description

Une table **MERGE** est un ensemble de tables **MyISAM** possédant la même structure ainsi que le même jeu de caractères.

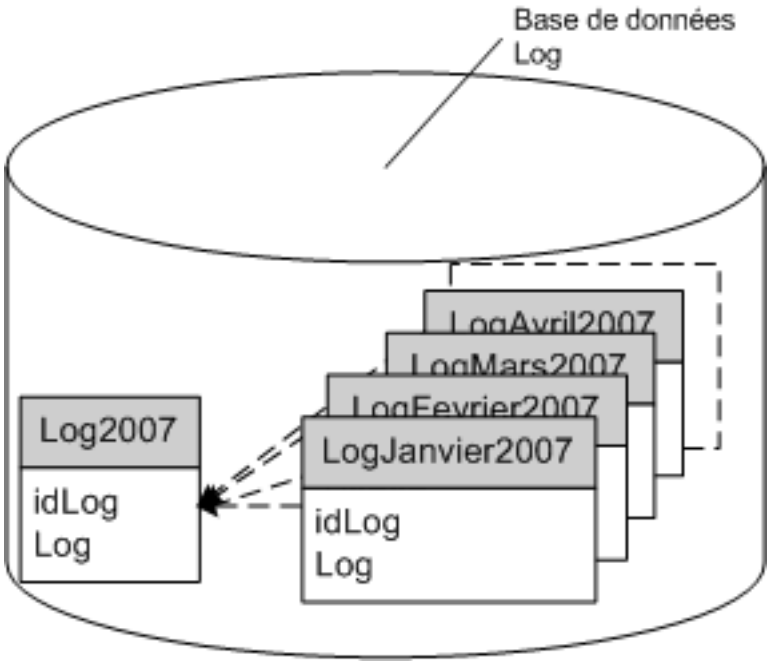
Il est comparable à l'opérateur ensembliste **UNION**, mais inclus dans le moteur lui-même et non pas dans une requête, ce qui garantit une meilleure optimisation. L'intérêt réside surtout dans le fait qu'une table MyISAM très volumineuse peut être coupée en plusieurs tables ou tables annexes.

L'efficacité est renforcée sur des tables dont les données sont compressées et en lecture seule.

L'intérêt peut être d'ordre stratégique. Imaginons qu'il existe une table par région. Des utilisateurs peuvent avoir les droits sur une table mais pas sur d'autres. Par contre, le **DBA** peut gérer toutes les tables des régions en une seule et ceci de manière complètement transparente pour les autres utilisateurs.

VI-B - Exemple d'utilisation

Partons d'un exemple où des logs sont entrés, à raison d'une table de logs par mois. Ces différentes tables (ou tables annexes) sont en lecture seule et peuvent être réunies en une seule grâce au moteur **MERGE**. Il est à noter aussi, que pour des opérations de maintenances, il est plus rapide de les exécuter sur plusieurs petits fichiers (une table **MyISAM** = un fichier) que sur un fichier qui possède un gros volume de données.



Structure d'utilisation Merge

Exemple d'utilisation

```
CREATE TABLE LogJanvier2007 (
    idlog INT NOT NULL AUTO_INCREMENT PRIMARY_KEY,
    log VARCHAR(255));

CREATE TABLE LogFevrier2007 (
    idlog INT NOT NULL AUTO_INCREMENT PRIMARY_KEY,
    log VARCHAR(255))

CREATE TABLE Log2007(
   idlog INT NOT NULL AUTO_INCREMENT PRIMARY_KEY,
   log VARCHAR(255))
TYPE = MERGE UNION=(LogJanvier2007, LogFevrier2007) INSERT_METHODE = LAST;
)

INSERT INTO LogJanvier2007(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 LogFevrier2007(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');
```

Selection sur la table de type Merge

```
SELECT * Log2007
-----------------------------------------------
|idLog | Log
| 1    | David;  10-01-2007 10:12:00;  External
| 2    | Sylvianne; 11-01-2007 14:12:54; External
| 3    | Albertus;15-01-2007 23:01:20; Internal
| 1    | Choulse;  14-02-2007 8:12:56; External
| 2    | Babacou; 20-02-2007 11:25:17; Internal
| 3    | Gorgio;20-02-2007 11:26:32; External
```

Il est à noter que les "identifiants uniques" ne le sont pas dans une table de type **MERGE**, mais le sont dans chacune des tables annexes. Les insertions de données (**INSERT**) sont possibles dans ce type de table, à condition de spécifier dans quelle table annexe les données doivent être insérées.

C'est possible grâce à l'option **INSERT_METHOD** spécifiée lors de la création de la table.

```
CREATE TABLE Log2007(
   idlog INT NOT NULL AUTO_INCREMENT PRIMARY_KEY,
   log VARCHAR(255))
TYPE = MERGE UNION=(LogJanvier2007, LogFevrier2007) INSERT_METHODE = LAST;);
```

Il y a 3 paramètres pour l'option INSERT_METHOD :

Les valeurs possibles sont :

- **NO** : Pour interdire toute création d'enregistrement via la table **MERGE**
- **FIRST** : Les enregistrements seront inscrits dans la première table définie dans la liste **UNION**.
- **LAST** : Les enregistrements seront inscrits dans la dernière table définie dans la liste **UNION**.

S'il est nécessaire d'insérer des enregistrements dans une table spécifique, alors l'opération devra être directement faite sur la table annexe.

VI-C - Fonctionnalités

Avantages

- Vu que les tables sont de type **MyISAM**, elles héritent des mêmes avantages
- La maintenance des tables annexes est facilitée par leur taille
- La mise en place (facile à faire et à défaire)
- **MERGE** permet de mettre en place un partitionnement horizontal pour les versions antérieures de MySQL inférieures à la 5.1. A partir de cette dernière version, MySQL prend en charge le partitionnement horizontal quel que soit le moteur de stockage.
- La rapidité d'exécution

Inconvénients

- L'obligation d'utiliser les tables de type **MyISAM** et donc les inconvénients de ce moteur
- Les index ne sont pas uniques.
- Le nombre élevé de fichiers sur la machine. Etant donné que le moteur **MERGE** utilise des tables de type **MyISAM**, chaque table annexe correspond à un fichier. S'il y a une table annexe par mois, il y aura donc 12 fichiers.
- La recherche par clés. Obligation de parcourir le catalogue de chaque table, ce qui ralentit le processus

VI-D - Conclusion

Le moteur **MERGE** permet d'administrer une base de données plus finement en termes de gestion de performances du serveur, en termes de droits sur les tables ainsi qu'en termes de maintenance sur les tables.
L'impossibilité d'annexion de tables se trouvant en dehors de la base de données, et ce même via une vue, est pénalisante.

VII - Le moteur Example

- **Licence :** GPL

- **Version de MySQL :** 4.1.3
- **Type :** Autre
- **Domaines d'application :**
  - Phase de développement de la structure SQL et de démonstration.

VII-A - Description

Ce type de table est assez particulier et ne sert pas à grand-chose, dans le sens où il n'est pas utile en production.

Ce type de table n'enregistre aucune donnée et seule la structure de la table existe sur le serveur (fichier **myTable.frm**). Il est très utile par contre, pour des tests dont la valeur des enregistrements n'intéresse pas, ainsi que pour fournir des exemples au niveau du code source de **MySQL** lors de la création d'un moteur de tables. Il est donc tout indiqué, comme cela a été dit précédemment, pour des développeurs et non pas pour de la production.

La grande utilité de ce type de table, se trouve dans les tests unitaires si la structure des tables est cohérente et si un objet gère les opérations d'écriture sur une table. S'il y a des erreurs de script **SQL** par rapport aux données qui auraient pu être injectées, elles seront remontées. Il est donc possible d'utiliser **SELECT**, **DELETE**, **INSERT** et **UPDATE**.
EXAMPLE peut servir à comprendre le fonctionnement d'un moteur de table car il est la représentation basique d'un moteur de table pour **MySQL**.

VII-B - Fonctionnalités

Avantages

- Ne met pas en danger l'intégrité des données de la base.
- Dédié aux tests unitaires

Inconvénients

- Ne gère pas les index (ce qui est normal vu qu'il n'y a pas de stockage de données).
- Les triggers ne fonctionnent pas.

VII-C - Conclusion

Il est aisé de comprendre que ce moteur est très utile lors des phases de développement, de tests unitaires ou de démonstrations. Il n'a donc pas sa place dans une base de données en production.

Il semblerait que ce moteur ne soit pas activé par défaut, du moins dans les versions **Win32**.

VIII - Le moteur BlackHole

- **Licence :** GPL
- **Version de MySQL :** 4.1.11
- **Type :** Autre
- **Domaines d'application :**
  - Phase de développement
  - Optimisation
  - Administration de la structure SQL

VIII-A - Description

Voici l'un des derniers-nés des moteurs de stockage.

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. Il est néanmoins possible d'utiliser toutes les commandes **SQL**, mais il faut bien comprendre qu'il ne créera et ne modifiera aucune donnée.

Alors, à quoi peut bien servir ce moteur ? Il y a plusieurs utilisations possibles, dont certaines restent non négligeables.

VIII-B - Exemples d'application

VIII-B-1 - Base de référence de performances

La première hypothèse consisterait à l'utiliser comme base de référence pour des tests de montée en charge.

Il faut pour cela, mesurer les performances avec le moteur **BLACKHOLE** sur le projet de base de données. Il faut ensuite effectuer les mêmes tests avec le moteur qui sera normalement utilisé en production (**MyISAM**, **InnoDB** etc.). En faisant une comparaison, il sera aisé d'obtenir un indice des performances de chaque moteur différent.

Ce fonctionnement peut être une bonne hypothèse d'utilisation.

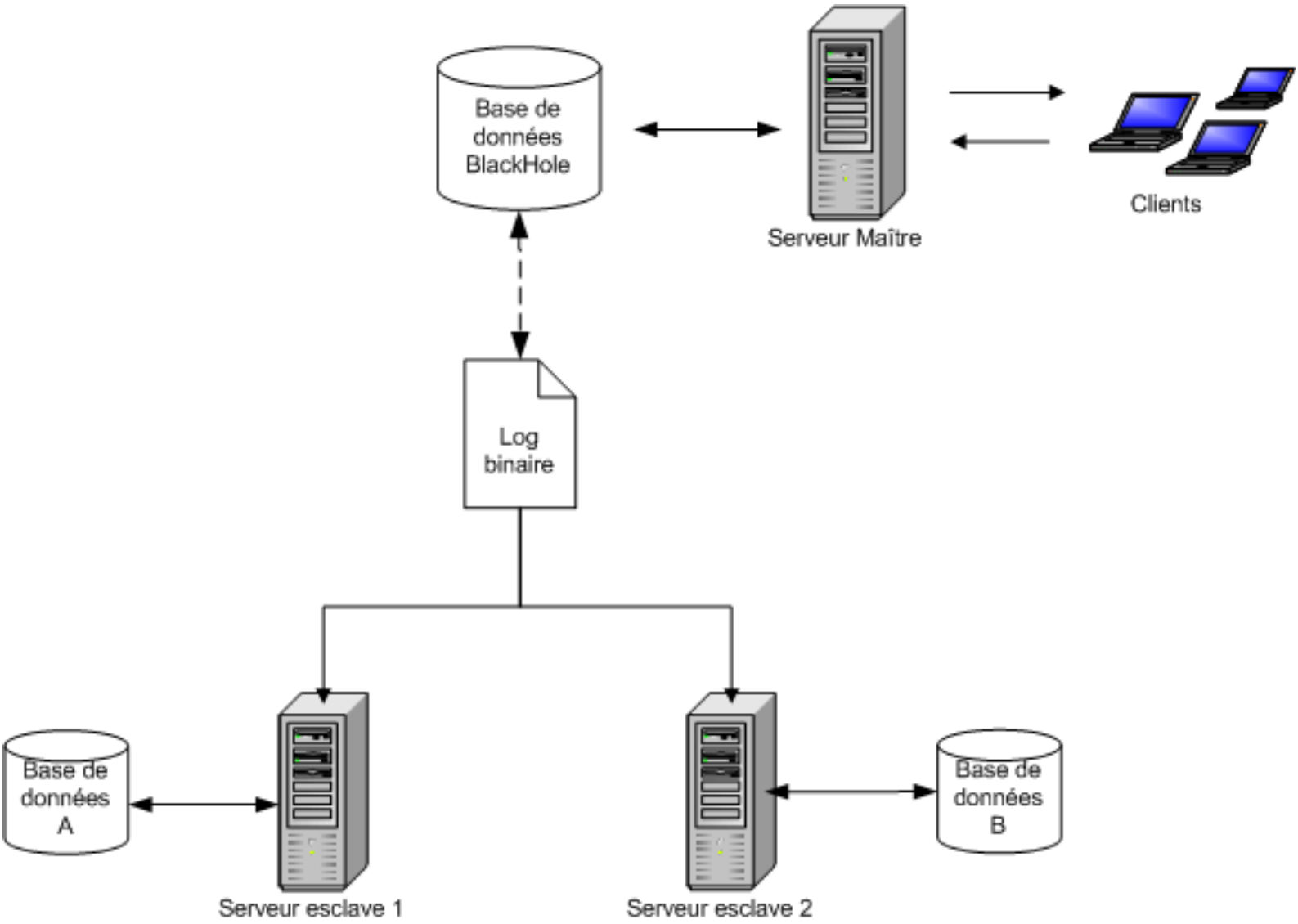
VIII-B-2 - Proxy MySQL

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**.

L'avantage certain est qu'il n'y aura pas de problème de synchronisation et pas, ou très peu, de temps de latence entre maître et esclave.



Utilisation du Proxy **MySQL** via le moteur **BLACKHOLE**.

VIII-B-3 - Déclencheur de cache de données

Une deuxième hypothèse serait l'utilisation du moteur **BLACKHOLE** comme déclencheur de cache de données **MySQL**.

MySQL peut stocker les données en mémoire afin d'augmenter les performances et réduire les accès disque. Pour ce faire, il faut charger toutes les données d'une table en mémoire via la commande **SELECT * FROM maTable** et injecter ces données dans une table de type **BLACKHOLE** ayant la même structure.

Déclencheur de cache
```
INSERT INTO maTableBlackHole SELECT * FROM maTable
```

Ceci aura pour effet de déclencher le cache de **MySQL** lié à la lecture ou l'écriture d'une table, même si elle est sans fond.

Il est à noter tout de même, que cette utilisation mérite pas mal de tests pour être sûr du gain de performance réalisé. De plus, il faut bien configurer le cache de requêtes dans le fichier de configuration de **MySQL**.

Je vous invite à lire cet article : **Étude pratique du cache de requêtes MySQL**.

VIII-B-4 - Recherche du coupable pour une opération gourmande

Lors d'une opération "*gourmande*" en CPU ou temps d'exécution, ce moteur peut s'avérer très utile. Il peut tester si votre script est en cause ou bien si le moteur choisi n'est pas adapté. Si lors du passage du script dans le moteur **BLACKHOLE** le temps d'exécution reste le même, alors le problème vient du script. Dans le cas contraire, cela vient peut-être d'un mauvais choix de moteur.

Ce moteur peut être très intéressant pour faire le choix d'un moteur approprié.

VIII-C - Fonctionnalités

Avantage

- Ne présente que des avantages pour ce qui a été présenté.

Inconvénients

- Peu de documentation.
- Faible disponibilité chez les hébergeurs.

VIII-D - Conclusion

Autour de ce moteur dont l'utilisation ne paraît pas claire à première vue, nous avons pu voir des domaines d'application où il peut se révéler très utile (maintenance, optimisation et administration).

Au vu de la documentation inexistante et des utilisations multiples possibles, il y a fort à parier que ce moteur fera parler de lui en ouvrant d'autres domaines d'application.

Il est regrettable qu'il ne soit pas plus répandu chez les hébergeurs.

IX - Le moteur FEDERATED

- **Licence :** GPL
- **Version de MySQL :** 5.0.3
- **Type :** Autre
- **Domaines d'application :** Administration de base de données

IX-A - Description

Le moteur **FEDERATED** permet de déporter les données sur un serveur distant.

Il est possible par exemple, d'avoir une base dont les données sont réparties sur plusieurs serveurs distants géographiquement. Dans une base utilisant le moteur **FEDERATED**, on ne trouve qu'un fichier ***.frm** qui est la définition de table. Les données sont stockées sur un ou plusieurs serveurs distants. Par le mot "distant", il faut comprendre que le serveur possédant une table **FEDERATED** devient client d'un autre serveur **MySQL**. Cela oblige cet autre serveur à se trouver à l'extérieur ou dans un réseau local. En aucun cas il ne peut être le "localhost".

Cela peut être presque comparable à une vue pour base de données **MySQL**.

IX-B - Exemple de mise en place d'une structure Federated

Sur le serveur "visible", se trouve une base de données contenant deux tables utilisant le moteur **FEDERATED** et pointant vers deux serveurs distants. Les serveurs **A** et **B** contiennent chacun une table de type quelconque (par exemple **MyISAM**). Dans ce type de structure, les moteurs utilisés par les serveurs distants n'ont aucune importance.

Syntaxe de 'connection' :

```
scheme://user_name[:password]@host_name[:port_num]:/db_name/tbl_name
```

Serveur visible

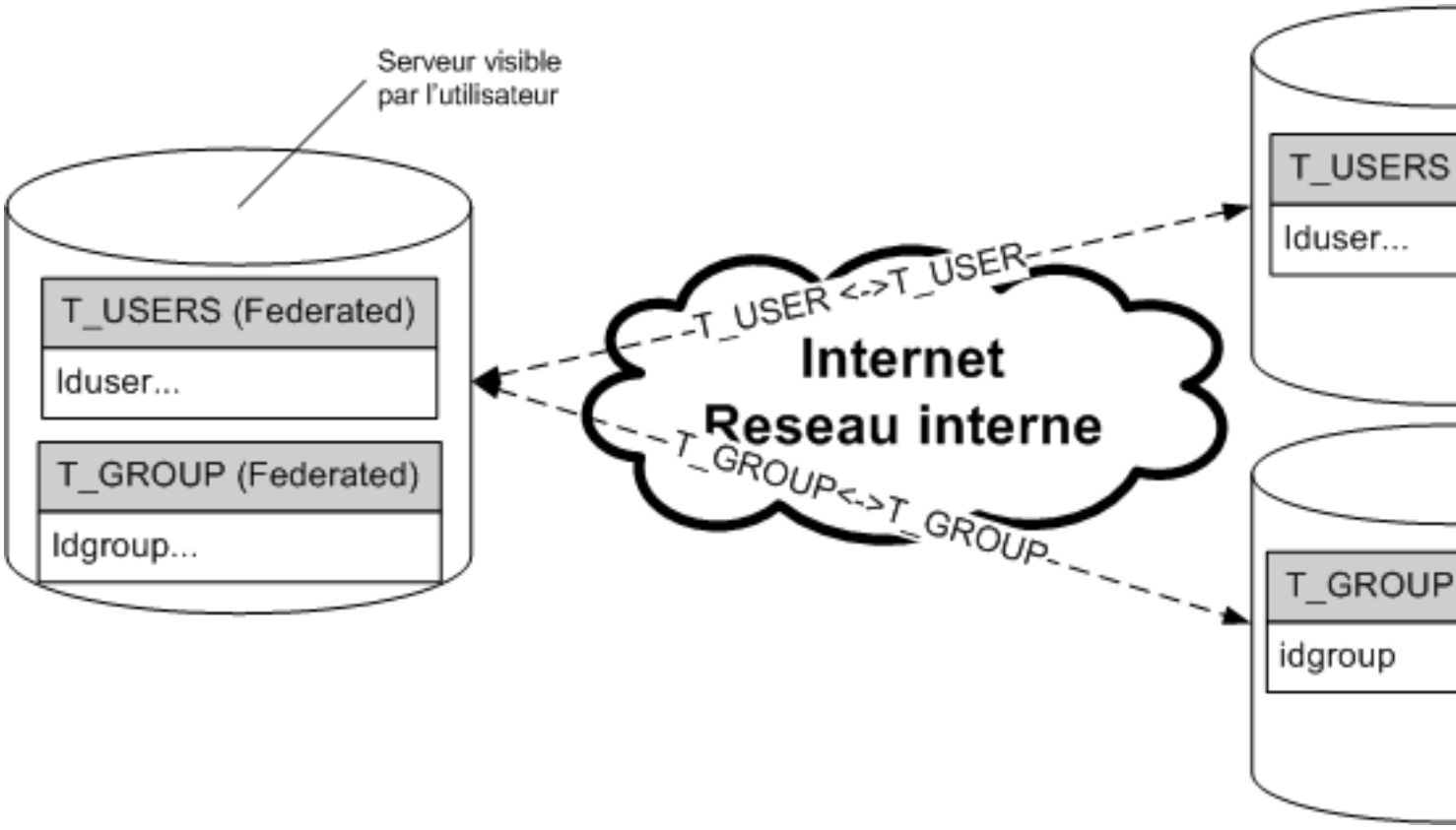
```
CREATE TABLE t_user (
    id      int(20) NOT NULL auto_increment,
    idgroup int(10) NOT NULL,
    name    varchar(32) NOT NULL default '',
    PRIMARY KEY  (id)
    KEY idgroup
)
ENGINE=Federated
CONNECTION='mysql://root@serverB:9306/t_user';


CREATE TABLE t_group (
    idgroup int(10) NOT NULL auto_increment,
    name    varchar(32) NOT NULL default '',
    PRIMARY KEY  (id)
)
ENGINE=Federated
CONNECTION='mysql://root@serverC:9306/t_group';
```

Serveur A

```
CREATE TABLE t_user (
    id      int(20) NOT NULL auto_increment,
    idgroup int(10) NOT NULL,
    name    varchar(32) NOT NULL default '',
    PRIMARY KEY  (id)
    KEY idgroup
) ENGINE=MyISAM;
```

Serveur B

```
CREATE TABLE t_group (
    idgroup int(10) NOT NULL auto_increment,
    name    varchar(32) NOT NULL default '',
    PRIMARY KEY  (id)
) ENGINE=MyISAM;
```



Dans les versions antérieures à **MySQL 5.1x**, la propriété contenant le paramétrage de connexion distante est **COMMENT** (mais elle ne servait pas pour autant à stocker des commentaires).

Quelques conseils

- Il est important d'avoir exactement la même structure entre la table de type **FEDERATED** et la table distante.
- Il est nécessaire de vérifier que le serveur distant est déjà activé avant d'y faire la moindre requête.
- Il est possible de pointer vers une table distante qui elle aussi utilise le moteur **FEDERATED**. Il faut être sûr de la cartographie du serveur distant et donc de faire très attention à ne pas tomber dans une boucle infinie.
- Si la structure de la table distante est différente de celle de la table **FEDERATED** attachée, le lien sera cassé.

IX-D - Fonctionnalités

Avantages

- Permet de répartir la charge.
- Pointer sur plusieurs bases de données distantes si elles acceptent les accès distants.
- Supporte les index.

Inconvénients

- Ne supporte que les commandes SQL simples (**SELECT**, **DELETE**, **INSERT**, **UPDATE**). Il n'est pas possible de faire **ALTER TABLE**, bien qu'il aurait été utile de pointer vers un autre serveur à la volée afin d'émuler un Load Balancing par exemple
- Ne gère pas les transactions.
- Les performances sont liées au réseau et à son débit.
- Le mot de passe apparaît en clair dans la propriété **COMMENT**.
- Pas de possibilité de savoir si des modifications ont eu lieu sur le serveur distant.

- Faible disponibilité chez les hébergeurs.

IX-C - Conclusion

Ce moteur offre de grandes opportunités d'administration d'une base de données.

Avec ce moteur, il est possible de faire une base de données métier à partir de plusieurs bases réparties géographiquement sur des machines distinctes.

Il est à noter que, lors de l'écriture de cet article, les tests tentés sur ce moteur sont restés vains.

X - Le moteur InnoDB

- **Licence :** GPL niveau 2. Société **InnoBase**, filiale depuis 2005 de la société Oracle.
- **Version de MySQL :** Par défaut depuis la version 4.0 de MySQL mais il y est possible de l'installer sur une version 3.23 de MySQL.
- **Type :** Transactionnel
- **Domaines d'application :** Application nécessitant une fiabilité de l'information avec une gestion des transactions

X-A - Description

InnoDB, est le moteur transactionnel le plus utilisé à l'heure actuelle dans les secteurs dit sensibles, c'est-à-dire nécessitant une cohérence et une grande intégrité des données.
Jusqu'à la version 5.1 incluse, c'est le seul moteur supportant les contraintes de clés étrangères (intégrité référentielle).

Il n'est pas concevable d'avoir des informations faisant référence à quelque chose d'inexistant. Peut-on imaginer un numéro de sécurité sociale qui ne soit pas associé à une personne ou un code postal associé à aucune ville ?
Il y a des domaines d'application où les données doivent être fiables à 100%.

Au-delà de l'intégrité référentielle, **InnoDB** propose des mécanismes transactionnelles présentant une grande compatibilité aux critères **ACID**.

X-B - Organisation interne

Avec une base de données composée de tables utilisant le moteur **InnoDB**, il est important de ne pas utiliser les mêmes méthodes qu'avec une base contenant uniquement des tables **MyISAM**.
Avec les tables utilisant le moteur **MyISAM**, il est facile de copier, supprimer une base de données : il suffit de copier le répertoire se trouvant dans le répertoire **/Data/** portant le même nom que la base de données.
De là, il est possible de le déplacer vers un autre serveur, de réaliser une autre base de donnés à partir de celle-ci, d'effectuer des sauvegardes.
Par contre, si la base de données comporte des tables utilisant le moteur **InnoDB**, il faudra faire plus attention.
En effet, toutes les données de toutes les tables de toutes les bases sont stockées dans un espace de tables commun.
De ce fait, la base devient un peu plus rigide.

X-C - Les clés étrangères

Exemple avec un moteur sans contrôle d'intégrité référentielle

```
CREATE TABLE t_user (
    iduser  int(20) NOT NULL auto_increment,
```

Exemple avec un moteur sans contrôle d'intégrité référentielle

```
    idgroup int(10) NOT NULL,
    name    varchar(32) NOT NULL default '',
    PRIMARY KEY  (id)
    KEY idgroup
)
ENGINE=MyISAM;

CREATE TABLE t_group (
    idgroup int(10) NOT NULL auto_increment,
    name    varchar(32) NOT NULL default '',
    PRIMARY KEY  (id)
)
ENGINE=MyISAM;

INSERT INTO t_group (name) VALUES ('invité');
INSERT INTO t_user (idgroup, name) VALUES (12, 'Charle');

SELECT * FROM t_user
-------------------------------------
iduser | idgroup | name
1      | 12      | Charle



SELECT * FROM t_group
-------------------------------------
idgroup | name
1       | invité

SELECT a.*, b.* FROM t_user AS a INNER JOIN t_user AS b ON a.idgroupe = b.idgroupe;
```

Cette dernière requête ne vous retournera aucun enregistrement, pour la simple raison que *Charles* n'est associé à aucun groupe existant dans la table **t_group**.
Dans cet exemple, vous observez bien qu'il y a une donnée fausse, du moins, qui ne fait référence à rien.

En effet, **idgroup** vaut 12 dans la table **t_user** mais elle n'existe pas dans la table **t_group**. La base de données n'est pas cohérente.
Nous pouvons dire "Il suffit de faire un **SELECT** pour vérifier l'existence de cet **ID**". Oui, c'est une solution, mais elle révèle que vous utilisez une astuce pour pallier à un problème. Si vous devez effectuer des tests à chaque requête, le serveur sera très sollicité.

Pour une société, cela n'est pas acceptable. Elle a besoin d'avoir confiance dans les données présentes, que le système soit cohérent.

Pour être sûr que l'utilisateur fait bien partie d'un groupe existant, nous devons utiliser une référence liée à la table **t_group** via le mot clé **FOREIGN KEY**, qui veut dire *clé étrangère*.
Il faut comprendre par l'utilisation d'un élément étranger à la table.

Exemple avec InnoDB

```
CREATE TABLE t_group (
    id INT(10) NOT NULL auto_increment,
    name    VARCHAR(32) NOT NULL default '',
    PRIMARY KEY  (id)
)
ENGINE=InnoDb;

CREATE TABLE t_user (
    iduser  INT(20) NOT NULL auto_increment,
    idgroup INT(10) NOT NULL,
    name    VARCHAR(32) NOT NULL default '',
    PRIMARY KEY  (id)
    KEY idgroup,
    FOREIGN KEY(idgroup) REFERENCES t_group(id)
)
```

Exemple avec InnoDB
```
 ENGINE=InnoDb;
 INSERT INTO t_group (name) VALUES ('webmaster');
 INSERT INTO t_user (idgroup, name) VALUES (LAST_INSERT_ID(), 'Leonardo');
```

Dans cet exemple, la table **t_user** est liée à la table **t_group** via le champ **idgroup**. Il n'est pas autorisé de placer un **idgroup** inexistant. Le champ **idgroup** est lié à la table **t_group**.

Vous ne pouvez pas référencer un champ n'existant pas encore.

X-D - Le COMMIT et ROLLBACK

Habituellement, lorsqu'une commande permettant de modifier les informations dans une table (**INSERT, UPDATE, DELETE**), il n'est plus possible de faire marche arrière.

Si vous estimez que votre programme est fiable à 100%, cela ne pose pas de problème. Mais nous ne sommes jamais à l'abri d'une erreur qui peut avoir un effet boule de neige sur toute la base de données. C'est pour cela qu'**InnoDB** intègre une notion d'annulation. En résumé, vous pouvez faire un **CTRL+Z** dans votre base de données.

Si nous reprenons l'exemple précédent avec la table t_group et t_user, nous allons placer volontairement un idgroup inexistant.

Exemple avec une erreur
```
SET @idGroup = 123;

CREATE TABLE t_group (
    id INT(10) NOT NULL auto_increment,
    name    VARCHAR(32) NOT NULL default '',
    PRIMARY KEY  (id)
) ENGINE=InnoDb;

CREATE TABLE t_user (
 iduser INT(20) NOT NULL auto_increment,
 idgroup INT(10) NOT NULL,
 name VARCHAR(32) NOT NULL default '',
 PRIMARY KEY (iduser),
 FOREIGN KEY (idgroup) REFERENCES t_group (id)
) ENGINE=InnoDb;


INSERT INTO t_group (name) VALUES ('webmaster');
INSERT INTO t_user (idgroup, name) VALUES (@idGroup, 'Leonardo');
```

Cette requête retourne une erreur car la valeur d'**idGroup** devrait valoir **1** et non pas **123**.
Par contre, le groupe est déjà existant, ce que nous ne souhaitions pas au début si cela fait partie d'une suite de processus.

Exemple avec avec gestion d'erreurs
```
DELETE FROM t_group;

SET @idGroup = 123;

SET AUTOCOMMIT = 0;

START TRANSACTION;

INSERT INTO t_group (name) VALUES ('webmaster');
Query OK, 1 row affected (0.00 sec)

INSERT INTO t_user (idgroup, name) VALUES (@idGroup, 'Leonardo');
```

Exemple avec avec gestion d'erreurs

```
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails
 (`test`.`t_user`, CONSTRAINT `t_user_ibfk_1` FOREIGN KEY (`idgroup`) REFERENCES `t_group` (`id`))

ROLLBACK;
```

Dans le code ci-dessus, il y a des mots clés ayant été ajoutés par rapport à l'exemple précédent.

- **AUTOCOMMIT = 0** : Dit à **MySQL** de ne pas valider les requêtes d'écriture.
- **START TRANSACTION** : Démarre la zone de la mise en tampon.
- **ROLLBACK:** : Donne l'ordre à MySQL de tout annuler depuis **START TRANSACTION**.

S'il n'y avait pas eu d'erreur, il aurait fallu faire un **COMMIT** au lieu du **ROLLBACK**. L'ensemble de la transaction aurait alors été validée.

Enfin, toute transaction ouverte et non validée (**COMMIT**) annulera l'opération à la déconnexion. **MySQL** effectuera un **ROLLBACK**.

X-E - Fonctionnalités

Avantages

- Verrouillage de ligne.
- Gestion du **COMMIT/ROLLBACK**
- Gère les gros volumes de données.
- Gestion des clés étrangères.
- Grande panoplie d'éléments de configuration du moteur.
- Gestion du backup sans bloquer une base en production.
- Couramment disponible chez les hébergeurs en mutualisé.

Inconvénients

- Lenteur de certaines opérations telles que le **SELECT COUNT(*) FROM maTable**.
- **TRUNCATE** n'est que le synonyme de **DELETE**.
- Les statistiques envoyées ne sont pas forcément précises : ce ne sont que des estimations.

X-F - Conclusion

Dans des domaines comme l'e-commerce, **InnoDB** saura répondre aux exigences en termes de fiabilité et de sécurité des données dans une base de données, dans la mesure où il sera utilisé avec les outils de transaction et d'intégrité référentielle. Sans cela, il sera préférable d'utiliser le moteur **MyISAM**.

XI - Le moteur BerkeleyBD (BDB)

- **Licence :** Sleepycat Software (Oracle)
- **Version de MySQL :** 3.23.34 mais n'est plus supporté à partir de la version 5.1 de Mysql
- **Type :** Transactionnel
- **Domaines d'application :** Applications à haute disponibilité

XI-A - Description

Le moteur **BDB** est là pour répondre à des exigences de certaines entreprises souhaitant avoir une haute sécurité sur leur données. En effet, ce moteur gère la transaction mais gère aussi le risque lié au crash d'un serveur.

À l'heure actuelle, sur les autres moteurs, si jamais votre serveur tombe, vous n'avez aucune possibilité de récupérer vos données sauf si vous avez la dernière sauvegarde. Mais combien de temps s'est-il passé entre la chute du serveur et la sauvegarde ?

Le moteur **BDB** possède un journal de transactions qui stocke les dernières opérations effectuées sur la table. Reportez-vous à la doc de **MySQL** pour connaître la procédure de récupération des données et la gestion des log binaires.

XI-B - Fonctionnalités

Avantages

- Les mêmes avantages que le moteur InnoDB.
- Gère le crash serveur.

Inconvénients

- N'est apparemment pas supporté sur un Windows et d'autres systèmes d'exploitation. Reportez-vous sur la documentation.
- Les même inconvénients que le moteur InnoDB.
- Disponibilité chez les hébergeurs.

XII - Le moteur Falcon

- **Licence :** MySQL AB
- **Version de MySQL :** 6
- **Type :** Transactionnel
- **Domaines d'application :** Application nécessitant une fiabilité de l'information avec une gestion des transactions au même titre que le moteur InnoDB.

XII-A - Description

Suite au rachat de la société **InnoSoft** par **Oracle**, **MySQL AB** a voulu répondre aux inquiétudes des personnes utilisant **InnoDB**. MySQL créa son propre moteur transactionnel sous le nom de **Falcon**.

Falcon sera disponible par défaut avec la version 6 de **MySQL**, mais il est déjà disponible en téléchargement sur le site de **MySQL**. **Falcon** n'a pas pour but de remplacer **InnoDB** immédiatement, car sa sortie est fréquemment reportée. MySQL AB n'a pas souhaité faire une copie d'**InnoDB** mais plutôt améliorer son utilisation et s'adapter aux nouvelles missions auxquelles **MySQL** fait face. D'après ses auteurs, Falcon répond aux nouvelles contraintes du **Web 2.0**, une gestion du crash beaucoup plus perfectionnée, gestion du crash serveur, une meilleure gestion des index de tables ...

Malgré ces bonnes volontés, une ombre plane au-dessus de Falcon. D'après des tests de performances, ce moteur serait très en retard par rapport à **InnoDB** et **MyISAM**.

Nous pouvons le mettre sur le compte d'un défaut de jeunesse, et **MySQL AB** travaille encore dessus afin d'améliorer les performances pour la sortie finale.

XIII - Le moteur Solid DB

- **Licence :** GNU GPL.
- **Version de MySQL :** 5.0.0
- **Type :** Transactionnel
- **Domaines d'application :**
  Applications critiques
- **Information :** **Site officiel de Solid DB For MySQL**

XIII-A - Description

Lors du rachat d'InnoDB par Oracle, ce moteur fut rappelé par **MySQL AB** pour être le remplaçant par intérim d'un moteur transactionnel libre.

SolidDB, la Ferrari de **MySQL**. D'après certains tests, c'est le moteur le plus performant en termes de rapidité d'exécution. Il est édité par la société Finlandaise **Solid Information Technology**. Selon ses auteurs, ce moteur est très sûr et répond parfaitement aux exigences des entreprises, surtout dans le domaine financier.

Les sociétés telles qu'**Alcatel**, **Nokia**, **Siemens**, **Nortel Networks**, **Cisco Systems** et **HP** utilisent ce moteur pour leurs applications critiques. Gage du sérieux de ce moteur.

- Accélération des performances dans un environnement mixte.
- L'intégrité des transactions des données est garantie grâce au support **ACID**.
- Capacité de configurer dynamiquement les paramètres.
- Diverses fonctions de sauvegarde en ligne.
- Détection des versions multiples.

XIII-B - Conclusion

Depuis le rachat d'**InnoDB** par **Oracle**, **SolidDB** est l'une des meilleurs alternatives dans le domaine du libre pour les moteurs transactionnels en attendant **Falcon**.

Il propose des performances garanties dans la gestion des domaines critiques. Au vu des grandes entreprises qui l'utilisent, ce moteur mérite une attention certaine.

L'inconvénient de **SolidDB** avec les versions antérieures à **MySQL 5.1** est qu'il exclut la présence des autres moteurs sur le serveur MySQL. Il n'est pas possible d'utiliser d'autres moteurs en même temps que **SolidDB**.

XIV - Le moteur PBXT (Prime Base XT)

- **Licence :** inconnue
- **Version de MySQL :** 4.1.21
- **Type :** Transactionnel
- **Domaines d'application :** Application ayant de fortes contrainte de disponibilité.

PBXT, un peu comme **SolidDB**, est un moteur promettant des performances élevées. Vous pouvez trouver plus de renseignements sur le **site de MySQL**.

XV - Le moteur AWS S3 (Simple Storage Service) Amazon S3

- **Licence :**Amazon
- **Version de MySQL :** 5.1
- **Type :** inconnu
- **Domaines d'application :** Service d'entreposage distant

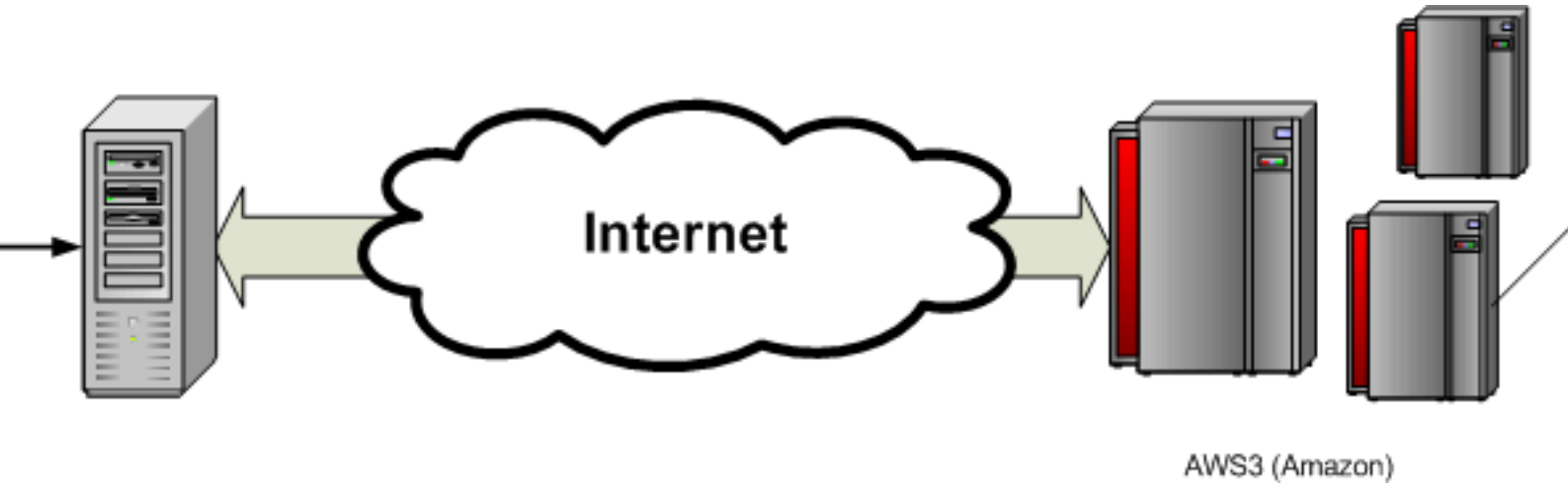
XV-A - Description

Le célèbre site de vente en ligne **Amazon** propose un service payant pour le stockage distant de données.

Il est fortement utilisé pour des sauvegardes en ligne à distance et non pas pour y accéder de manière continue. Via ce service, il est possible de gérer de gros volumes de données. Amazon S3 vous débarrasse des contraintes liées à la machine, à son emplacement, au type de serveur *etc*. Grâce à ce service, vous pouvez avoir une hébergement quasiment illimité pour vos données.

Autre point, vous ne payez que ce que vous consommez.

*Pour plus de détails : **le site d'Amazon S3**.*



XV-B - Conclusion

Ce genre de service est assez original dans le sens où il vous débarrasse de tous les problèmes matériels. **Amazon** propose un service clé en main pour les besoins d'entreprises ne souhaitant pas investir en personnel et en matériel.

XVI - Le moteur NitroDB for MySQL

- **Licence :** NitroSecurity
- **Version de MySQL :** Inconnue
- **Type :** Transactionnel
- **Domaines d'application :** Entreposage et temps de réponse sur de forts volumes de données.
- **Information :** **Site officiel de NitroSecurity**

XVI-A - Description

Le moteur BrightHouse est de type relationnel à haut débit. Dans le domaine d'entreposage de forts volumes de données, ce moteur répondra aux exigences les plus pointues en terme de temps de réponse.

Pas besoin de chercher dans votre serveur de base de données la présence de ce moteur, il ne s'y trouve pas par défaut.

XVII - Le moteur BrightHouse for MySQL

- **Licence :** Infobright Inc.
- **Version de MySQL :** Inconnue
- **Type :** Non transactionnel
- **Domaines d'application :** Administration

XVII-A - Description

À l'heure actuelle, **BrightHouse** for **MySQL** est un moteur comme le moteur **Archive** à la différence qu'il a une meilleur compression (Rapport moyen 10:1, au maximum 30:1). Il gère de très gros volumes de données (multitéra-octet).

XVIII - Le moteur Maria

- **Licence :** MySQLAB.
- **Version de MySQL :** 6.0
- **Type :** Non transactionnel
- **Domaines d'application :** De par son approche, il hérite du même domaine d'application qu'InnoDB.

XVIII-A - Description

Le moteur **Maria**, dernier-né des moteurs pour **MySQL**, est un clone du moteur **MyISAM** à la différence que ce moteur gère le crash serveur. Ce moteur est livré dans une installation spécifique sur la version 5.1 de **MySQL**.

XIX - Conclusion

Au travers de cet article, vous avez pu voir l'efficience de l'API **MySQL** pour l'intégration des moteurs de tables.
Il faut noter deux groupes de moteurs : l'un pour les phases de développement (**BlackHole**, **Example**) ; l'autre pour la production.

Ainsi, avec ses différents moteurs de tables, **MySQL** peut couvrir vos exigences dans vos projets de bases de données. Certes, l'article n'est pas complet, mais il sera mis à jour en fonction des nouvelles concernant les moteurs de stockage. En effet, les moteurs **Falcon** et **Maria** feront beaucoup parler d'eux et nous ne manquerons pas d'y revenir dans cet article.

De ce fait, **MySQL** montre qu'il a sa place parmi les grands.

À noter que si vous avez des informations à fournir ou à compléter, n'hésitez pas à intervenir.

XX - Remerciements

J'ai presque honte de le dire, mais voilà près d'un an que cet article est en chantier et pour cela je tenais à remercier **Yogui** pour sa patience et pour ses conseils concernant le montage de cet article. Je remercie aussi **Antoun** et **Yogui** pour le temps passé pour la correction et les conseils concernant ce sujet.