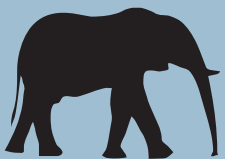
MySQL 5

mémento

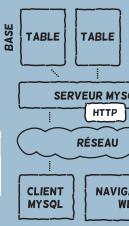
Rodolphe **Rimelé**

4e édition



EYROLLES

Connexion à une base de données



Étape préliminaire avant toute opération : la connexion à la base se fait via un client **mysql**, en ligne de commande ou en mode graphique, ou encore à travers un langage de programmation (PHP, C, Ruby, Python...).

Connexion en ligne de commande avec le client MySQL

Sous Linux/Unix, Mac OS X, et sous Windows :

`mysql -u`*login* `-p`*mot_de_passe*	Connexion sans spécifier de base de données
`mysql -u`*login* `-p`*mot_de_passe* *nom_base*	Connexion spécifiant une base de données

Une fois connecté, saisir directement les requêtes closes d'un point-virgule.

Productivité : instructions utiles en ligne de commande MySQL

`source` *nom_de_fichier*	Exécute les commandes SQL stockées dans un fichier.
`quit` **OU** `exit`	Quitte le client MySQL.
`use`	Choisit la base de données à utiliser.
`status`	Affiche l'état du serveur.
`?`	Affiche l'aide.
`tee` *nom_de_fichier*	Enregistre les commandes entrées et leurs résultats dans un fichier journal.

Autres commandes shell utiles

`mysqldump`	Exporte vers un fichier texte pour sauvegarde ou transfert entre serveurs, une (`--opt nom_base`) ou toutes (`--all-databases`) les bases. EXEMPLE `mysqldump --opt` *nom_base* `>` *base.sql* `mysqldump --all-databases >` *bases.sql*
`myisamchk`	Vérifie, optimise et répare les tables au format MyISAM. ATTENTION Le serveur doit être stoppé durant les opérations.
`mysqlimport`	Importe un fichier texte dans une table (similaire à `LOAD DATA INFILE`).
`mysqladmin`	Permet de créer/supprimer une base, exporter des tables, recharger les droits d'accès, etc.
EXEMPLE	Pour changer le mot de passe root `mysqladmin -u` *root* `password` *nouveau_mot_de_passe*
`mysqlcheck`	Utilitaire client de maintenance des tables, pour les vérifier, réparer, analyser et optimiser. ATTENTION Le serveur doit être démarré durant les opérations.
`mysqlshow`	Affiche des informations sur une table ou un index.
`mysqldumpslow`	Affiche le journal des requêtes les plus lentes, si l'option `log-slow-queries` est activée sur le serveur.

Connexion via PHP

Après avoir établi la connexion, la fonction **query** transmet la requête au serveur. Diverses fonctions permettent de lire les résultats retournés.

Avec PDO (PHP version 5.3 et supérieures)

```
$sqlpdo = new PDO("mysql:host=$mysql_server;dbname=$mysql_base",
$mysql_login, $mysql_password);
$result = $sqlpdo->query('SELECT * FROM table');
foreach ($result as $ligne) {
  echo $ligne['champ']; }
```

Avec l'API MySQLi (PHP version 5 et supérieures)

```
mysqli_connect($mysql_server,$mysql_login,$mysql_password);
mysqli_select_db($mysql_base);
$result = mysqli_query('SELECT * FROM table');
while($ligne = mysqli_fetch_assoc($result))
  echo $ligne['champ'];
mysqli_close();
```

Avec MySQLi en mode objet

```
$sql = new mysqli($mysql_server,$mysql_login,$mysql_password,$mysql_base);
$result = $sql->query('SELECT * FROM table');
while($ligne = $result->fetch_assoc()) echo $ligne['champ'];
$sql->close();
```

Utiliser une base

En ligne de commande, renseignent sur le contenu disponible (bases et tables).

`SHOW DATABASES;`	Liste les bases disponibles
`USE` *nom_de_la_base*`;`	Définit la base active à utiliser
`SHOW TABLES;`	Liste les tables présentes dans une base
`DESCRIBE` *nom_de_la_table*`;`	Décrit la structure d'une table (champs)

Fonctions courantes

`mysqli_query($requete)`	Exécute une requête et renvoie le résultat.
`mysqli_fetch_row($result)`	Renvoie la ligne dans une liste.
EXEMPLE `while(list($champ1,$champ2) = mysqli_fetch_row($result)) { ... }`	
`mysqli_fetch_assoc($result)`	Renvoie la ligne dans un tableau associatif.
`mysqli_fetch_array($result)`	Renvoie la ligne dans un tableau.
`mysqli_affected_rows()`	Renvoie le nombre de lignes affectées par la dernière commande.
`mysqli_insert_id()`	Renvoie l'identifiant unique (EXEMPLE **PRIMARY KEY** en **AUTO_INCREMENT**) de la dernière commande INSERT.
`mysqli_error()`	Renvoie le message d'erreur né de la dernière requête.
`mysqli_num_rows($result)`	Renvoie le nombre de lignes d'un résultat.
`mysqli_close()`	Ferme la connexion MySQL.

ATTENTION Le préfixe mysqli est à remplacer par **mysql** dans ces noms de fonctions si le serveur n'est pas équipé de MySQLi. En outre, en mode objet, on omet le préfixe si la fonction est appelée en tant que méthode de l'instance utilisée (EXEMPLE **$mysqli->fetch_assoc(), $mysqli->query($requete)**) ou en tant qu'attribut dans le cas de valeurs simples (EX. **$result->num_rows, $sql->error**).

Création d'une base et de tables

Un serveur MySQL contient une ou plusieurs bases de données, interrogeables par des requêtes. Chaque base contient au moins une table, structurée en champs (colonnes) définis à sa création, et contenant zéro à plusieurs enregistrements (lignes).

REMARQUE Ce mémento offre une sélection des commandes les plus utiles, n'hésitez pas à consulter la documentation officielle sur http://dev.mysql.com/doc/.

Créer une nouvelle base de données ou une nouvelle table

```
CREATE DATABASE nom_de_la_base;
CREATE TABLE nom_de_la_table (
  nom_champ1 type_champ1(longueur) attributs,
  nom_champ2 type_champ2(longueur) attributs, ...
  type_clé1 nom_clé1, type_clé2 nom_clé2, ...
) TYPE = type_de_table;
```

Chaque champ (colonne) est défini par un nom, un type dont la longueur est spécifiée (voir plus loin « Types de champs »), et des attributs, séparés par des virgules, pouvant prendre ces valeurs :

NULL / NOT NULL	Le champ peut être vide, prendre ou pas la valeur NULL.
DEFAULT *'valeur_par_defaut'*	Le champ a une valeur par défaut automatique si elle n'est pas précisée lors d'une insertion.
AUTO_INCREMENT	Sa valeur est auto-incrémentée à chaque nouvelle insertion (clé primaire et type numérique requis).

REMARQUE La longueur est facultative pour les types numériques, CHAR et VARCHAR. Elle n'est pas permise pour les autres types (dates, TEXT, BLOB, ENUM, SET...).

Les clés (index) de la table sont déclarées par leur type (voir plus loin « Types d'index ») et le(s) champ(s) à indexer.
Le type de table est spécifié à la création par **TYPE=*type_de_table*** en fin d'instruction **CREATE TABLE**. Les types par défaut sont :

MyISAM	Format par défaut
MEMORY	**(HEAP)** En mémoire vive, rapide mais perte des données au redémarrage du serveur
InnoDB	Permet les transactions, les clés étrangères et les locks avancés

EXEMPLE

```
CREATE TABLE adresses (id INT NOT NULL UNSIGNED AUTO_INCREMENT,
nom CHAR(50) NOT NULL, prenom CHAR(40), anniversaire DATE,
PRIMARY KEY(id)
) TYPE = MyISAM;
```

Créer une table ayant la même structure qu'une autre

```
CREATE TABLE nouvelle_table LIKE ancienne_table;
```

Créer un index

```
CREATE [type_index] INDEX nom_index ON nom_table (col_indexee,...);
```

Met en place une clé ***nom_index*** de type ***type_index*** (**UNIQUE**, **FULLTEXT** ou **SPATIAL**) sur un ou plusieurs champs, dont les noms sont séparés par des virgules.

PERFORMANCES Il est possible de spécifier la longueur du champ à indexer entre parenthèses, notamment lorsqu'il s'agit de champs texte et il est souvent inutile d'indexer toute la chaîne de caractères pour des raisons de performances.
EXEMPLE : **CREATE INDEX** *debut_prenom* **ON** *carnet_adresses*(prenom(*5*));


Renommer/supprimer bases, tables et clés

`RENAME TABLE` *ancien_nom* `TO` *nouveau_nom*; **Remarque** Nécessite les privilèges ALTER et DROP sur l'ancienne table, CREATE et INSERT sur la nouvelle.	Renomme une table
`DROP TABLE` *nom_de_la_table*, *nom_de_la_table2*,...;	Supprime une ou plusieurs tables
`DROP DATABASE` *nom_de_la_base*;	Supprime une base de données
`DROP INDEX` *nom_index* `ON` *nom_table*;	Supprime un index

Types de champs

Performances Le choix du type de champ influe sur les performances et l'espace de stockage, ainsi que sur la simplicité des requêtes qui en découleront. Il définit un nombre fixe ou variable d'octets pour le stockage de chaque valeur.

*INT pour le stockage de nombres entiers

Type	Nb d'octets	De...	à...
`TINYINT`	1	-128	127
`SMALLINT`	2	-32 768	32 767
`MEDIUMINT`	3	-8 388 608	8 388 607
`INT`	4	-2 147 483 648	2 147 483 647
`BIGINT`	8	-9 223 372 036 854 775 808	9 223 372 036 854 775 807

Remarques Si le mot-clé UNSIGNED est spécifié, la limite inférieure est 0 (seuls des nombres positifs sont stockés), et la limite supérieure est doublée car on s'affranchit du signe (ex. TINYINT UNSIGNED : 0 à 255). INTEGER est un alias pour INT. On peut spécifier la longueur maximale du champ, entre parenthèses, après le type.

Exemples
age `TINYINT(3) UNSIGNED NOT NULL,`
code_postal `MEDIUMINT(5) UNSIGNED NOT NULL`

L'attribut ZEROFILL définit le caractère « 0 » comme remplissage par défaut. **Exemple** en stockant la valeur 3 dans un champ `INT(4) ZEROFILL`, la valeur lue sera 0003.

BIT, BOOL, BOOLEAN : types booléens pour valeurs binaires

Expriment une variable ne prenant que deux valeurs (vrai ou faux). Équivalent à un `TINYINT(1)` stockant `1` ou `0` dans un champ numérique.

FLOAT, DOUBLE, DECIMAL pour les nombres à virgule flottante

`FLOAT` et `DOUBLE` (équiv. `REAL`) stockent des nombres à virgule flottante. `DECIMAL` (équiv. `NUMERIC`) stocke sous forme littérale : chaque caractère représente un chiffre stocké.

Remarque Les attributs UNSIGNED et ZEROFILL sont là aussi applicables.

CHAR et VARCHAR pour les chaînes de caractères

Stockent des chaînes sur un nombre maximal d'octets indiqué entre parenthèses – les chaînes excédant ce maximum sont tronquées. Si la chaîne stockée occupe moins d'espace, `CHAR` le réserve quand même en complétant de blancs, à la différence de `VARCHAR` qui ne stocke que la chaîne utile (mais est un peu moins performant).

Remarque Si une table comporte au moins un champ de taille variable (VARCHAR, TEXT, BLOB...), tous les champs CHAR sont automatiquement convertis en VARCHAR à la création de la table, car il n'y a aucun intérêt à les laisser en taille fixe.

BINARY et VARBINARY pour les chaînes binaires

Diffèrent de la même façon que `CHAR` et `VARCHAR` (voir ci-avant). Ils n'ont pas de jeu de caractères associé, les tris et comparaisons sont basés sur la valeur numérique de l'octet, et on spécifie leur taille maximale de la même façon que `CHAR` et `VARCHAR`, à la différence près qu'elle est définie en octets.

TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT, BLOB, TINYBLOB, MEDIUMBLOB, LONGBLOB pour les chaînes longues

Une colonne de type `TEXT` est assimilable à un `VARCHAR` de grande capacité. Ce type peut contenir une quantité variable de données, en fonction de l'un des 4 types. Les types `BLOB` (*Binary Long Object*) diffèrent des types `TEXT` : ils sont sensibles à la casse (majuscules/minuscules) et triés sur leur valeur binaire.

Remarque On ne peut spécifier de valeur par défaut pour les types BLOB et TEXT.

Types de champs

Type	Capacité de stockage en caractères
TINYTEXT / TINYBLOB	255
TEXT / BLOB	65 535
MEDIUMTEXT / MEDIUMBLOB	16 777 215
LONGTEXT / LONGBLOB	4 294 967 295 (4 Go)

Exemples

```
prix DECIMAL(6,2), poids FLOAT UNSIGNED, nom CHAR(30), prenom
CHAR(50),
article TEXT NOT NULL,
```

ENUM et SET pour proposer une liste de choix

ENUM permet le choix d'une valeur parmi une liste définie à la création de la table. Il est possible d'utiliser une chaîne vide (' '). La valeur par défaut sera le premier élément de l'énumération si la colonne est déclarée **NOT NULL**, ou **NULL** si la colonne dispose de cet attribut. Une énumération peut avoir un maximum de 65 535 éléments. **SET** diffère d'**ENUM** car il peut avoir 0 ou *plusieurs* valeurs choisies dans la liste définie à la création de la table. Les valeurs sont séparées par des virgules.

Remarque Pour sélectionner des champs selon des valeurs de SET, le plus simple est d'utiliser la fonction **FIND_IN_SET**.

Exemple

```
couleur ENUM('bleu','vert','indigo'),
colis SET('lourd','fragile','rond'),
SELECT * FROM nom_table WHERE
FIND_IN_SET('fragile',colis)>0;
```

Gestion des dates avec TIME, DATETIME, DATE, TIMESTAMP, YEAR

Type	Format	De	A
YEAR	AAAA	1901	2155
DATE	AAAA-MM-JJ	1000-01-01	9999-12-31
DATETIME	AAAA-MM-JJ HH:MM:SS	1000-01-01 00:00:00	9999-12-31 23:59:59
TIME	HH:MM:SS	-838:59:59	838:59:59
TIMESTAMP	AAAA-MM-JJ HH:MM:SS	1000-01-01 00:00:00	9999-12-31 23:59:59

Le type **TIMESTAMP** diffère des autres types date, car il stocke automatiquement l'heure courante lors d'une commande **INSERT** ou **UPDATE** sur les enregistrements affectés. En d'autres termes, MySQL applique automatiquement la fonction **NOW()** lorsqu'aucune valeur par défaut n'est spécifiée lors de l'insertion ou de la modification.

Remarque Pour empêcher la mise à jour ultérieure du champ lors de la modification du champ, lui affecter sa propre valeur :
UPDATE *nom_table* SET *colonne_timestamp=colonne_timestamp, ...;*

Exemples

```
UPDATE rendezvous SET heure='14:30:20' WHERE contact_id=3;
INSERT INTO agenda (description,jour) VALUES ('KiwiParty','2014-06-13');
annee YEAR NOT NULL,
SELECT * FROM nom_table WHERE annee=2014;
```

Types d'index (clés)

Pour la bonne performance d'une base de données à travers le temps et à mesure de son remplissage, il est souvent nécessaire d'y adjoindre un ou plusieurs index accélérant la recherche. Rappelons qu'un index est employé dans une clause **WHERE**, que tout type de colonne peut être indexé, qu'il se crée avant ou après la création de la table (voir **CREATE INDEX**) et que **KEY** est synonyme de **INDEX**.

Exemple

```
CREATE TABLE livres (id INT UNSIGNED, titre CHAR(50), editeur CHAR(100), descrip-
tion TEXT, INDEX idx_titre (titre) INDEX idx_titre_editeur (titre,editeur) );
```

- **PRIMARY KEY** définit la clé primaire d'une table SQL et porte aussi sur un ou plusieurs champs, pour identifier de façon unique et rapide un enregistrement précis (ex. : numéro de sécurité sociale). **Exemple** **PRIMARY KEY** (*id*)
- **UNIQUE** n'autorise que l'insertion de valeurs ne figurant pas déjà dans la colonne. Il interdit les doublons car rend impossible d'obtenir deux enregistrements pour lesquels le champ est strictement identique. **Exemple** **UNIQUE KEY** (*titre*)
- **FULLTEXT** est voué à l'indexation de texte et permet la recherche d'un ou plusieurs mots avec évaluation de la pertinence (voir "Requêtes avancées", **FULLTEXT** et **MATCH ... AGAINST**). N'indexe que des **CHAR**, **VARCHAR** ou **TEXT** de tables **MyISAM** ou **InnoDB**. **Exemple** **FULLTEXT KEY** (*description*)



Gestion des utilisateurs

`CREATE USER` *login* `IDENTIFIED BY '`*mot_de_passe*`';`	Crée un nouvel utilisateur
`RENAME USER` *ancien_login* `TO` *nouveau_login*;	Renomme un utilisateur
`SET PASSWORD FOR` *login* `= PASSWORD('`*nouveau_mot_de_passe*`');`	Change un mot de passe
`DROP USER` *login*;	Supprime un utilisateur
`GRANT` *droits* `ON` *cible* `TO` *login* `WITH` *options*`;`	Modifie ou attribue des droits à un utilisateur
Remarque Voir ci-après le détail des options pour la gestion des droits.	
`REVOKE` *droits* `ON` *cible* `FROM` *login*`;`	Révoque les droits d'un utilisateur
`FLUSH PRIVILEGES;`	Recharge les droits MySQL
Attention N'oubliez pas de recharger les droits après toute modification des droits utilisateur !	

Remarque **Options pour la gestion des droits**

`cible` : nom d'une table *`nom_table`* (nom complet : *`nom_base.nom_table`*). Pour attribuer les droits à l'ensemble des tables d'une base : *`nom_base.*`*

`login` : un login ou plusieurs séparés par des virgules, avec éventuellement le mot de passe : `TO '`*`login`*`'@'`*`localhost`*`' IDENTIFIED BY '`*`mot_de_passe`*`'`.

`ALL`	Tous les droits
`ALTER`	Autorise `ALTER TABLE`
`CREATE`	Autorise `CREATE TABLE`
`CREATE VIEW`	Autorise `CREATE VIEW`
`DELETE`	Autorise `DELETE`
`DROP`	Autorise `DROP TABLE`
`EXECUTE`	Autorise l'emploi de procédures stockées
`FILE`	Autorise `SELECT ... INTO OUTFILE` et `LOAD DATA INFILE`
`INDEX`	Autorise `CREATE INDEX` et `DROP INDEX`
`INSERT`	Autorise `INSERT`
`LOCK TABLES`	Autorise `LOCK TABLES`
`PROCESS`	Autorise `SHOW FULL PROCESSLIST`
`RELOAD`	Autorise `FLUSH`
`SELECT`	Autorise `SELECT`
`SHOW DATABASES`	Autorise `SHOW DATABASES`
`SHUTDOWN`	Autorise mysqladmin shutdown
`SUPER`	Autorise entre autres `KILL` et `SET GLOBAL`
`UPDATE`	Autorise `UPDATE`
`USAGE`	Synonyme de "pas de privilèges"
`GRANT OPTION`	Synonyme de `WITH GRANT OPTION` (voir ci-dessous)

WITH `GRANT OPTION`	Donne le droit de modifier les droits des autres utilisateurs
WITH `MAX_QUERIES_PER_HOUR` *n*	N'autorise que *n* requêtes par heure
WITH `MAX_UPDATES_PER_HOUR` *n*	N'autorise que *n* requêtes `UPDATE` par heure
WITH `MAX_CONNECTIONS_PER_HOUR` *n*	N'autorise que *n* connexions par heure
WITH `MAX_USER_CONNECTIONS` *n*	N'autorise que *n* connexions simultanées

Exemple

`GRANT SELECT,UPDATE,DELETE,INSERT ON`*`mabase`*`.* TO '`*`monlogin`*`'@'`*`localhost`*`' IDENTIFIED BY '`*`motdepassedifficileàtrouver`*`';`

Remarque `localhost` ne tolère que les connexions locales sur le serveur, `%` tolère toutes les sources externes (à utiliser avec précaution).

Modifier la structure d'une table

Pour changer la structure d'une table existante : `ALTER TABLE` *`nom_table ...`*
Suivi, au choix, des instructions suivantes :

`RENAME` *nouveau_nom_table*	Renomme la table
`MODIFY` *nom_champ nouveau_type*	Modifie le type d'un champ
`CHANGE` *nom_champ nouveau_nom_champ nouveau_type*	Change le champ *nom_champ* vers *nouveau_nom_champ* avec le type *nouveau_type*
`ADD` *nom_champ type_champ*	Ajoute un champ
Exemple `ALTER TABLE` *table* `ADD` *code_postal* `CHAR(5) AFTER` *ville*; Remarque On précisera la position du champ dans la structure avec les clauses `AFTER` (après un autre champ à préciser) et `FIRST` (en premier).	
`ADD INDEX` *nom_index type_index* (*champ, champ2...*)	Ajoute un index de type *type_index* sur *champ, champ2...*
`ADD PRIMARY KEY` *nom_index type_index* (*champ, champ2...*)	Ajoute une clé primaire de type *type_index* sur *champ, champ2...*

Modifier la structure d'une table

`ADD UNIQUE nom_index type_index (champ, champ2...)`	Ajoute un index unique sur champ de type *type_index* sur *champ, champ2...*
`ADD FULLTEXT nom_index (champ, champ2 ...)`	Ajoute un index de type `FULLTEXT` sur *champ, champ2...*
`DROP nom_champ`	Supprime le champ de la structure de la table.
`DROP PRIMARY KEY`	Supprime la primaire de la structure de la table.
`DROP INDEX nom_index`	Supprime un index de la structure de la table.

Requêtes

`SELECT champ1,champ2 FROM nom_table; clause_where clause_group clause_order clause_limit;`	Sélectionne tous (`*`) ou certains enregistrements d'une table, répondant éventuellement à des conditions.

Clauses conditionnelles Voir plus loin les clauses WHERE, GROUP BY, ORDER BY, LIMIT pour les requêtes conditionnelles et triées.

`SELECT nom,ROUND (2014-annee_naissance) AS age FROM adresses ORDER BY age DESC;`	Le mot clé `AS` crée un alias sur un nom de colonne ou de table. Il est ainsi possible de raccourcir les noms utilisés pour des commodités d'écriture. (Ci-contre alias sur champ puis sur table)
`SELECT a.jour,p.nom FROM anniversaires AS a,carnet_adresses AS p WHERE a.id = p.id;`	
`INSERT INTO nom_table (champ1,champ2,...) VALUES ('valeur1','valeur2',...);`	Insère de nouveaux enregistrements dans une table.

Remarques Si une colonne n'est pas spécifiée dans la liste, elle prend la valeur par défaut qui lui a été assignée lors de la création de la table. Si une colonne possède l'attribut `AUTO_INCREMENT` il suffit de ne pas préciser la colonne à l'insertion, ou de lui donner la valeur `NULL`. `INSERT` permet les insertions multiples : INSERT INTO *mementos* (*titre,editeur*) VALUES ('*MySQL*','*Eyrolles*'),('*HTML5*','*Eyrolles*');

`DELAYED`	En précisant la clause `DELAYED` après `INSERT`, le serveur rend la main au client et n'insère les lignes que lorsque la table est inutilisée.
`LOW_PRIORITY`	La clause `LOW_PRIORITY` retarde l'insertion jusqu'à ce qu'il n'y ait plus de client lisant la table, mais ne rend pas la main.
`INSERT INTO mementos (titre_memento,description_memento, prix_memento) SELECT titre,description, '5' FROM livres WHERE type='memento';`	`INSERT... SELECT` permet d'insérer des valeurs à partir d'une ou plusieurs autres tables grâce à une sous-requête `SELECT`.
`UPDATE nom_table SET champ1= 'nouvelle_valeur',champ2='autre_valeur' WHERE champ3=champ2+7;`	Met à jour des enregistrements dans une table, concernés ou non par une condition. L'`UPDATE` peut porter sur plusieurs tables.

Exemple `UPDATE produit,catalogue SET produit.prix=catalogue.prix WHERE produit.id=catalogue.id;`

Remarque Il est possible d'utiliser la clause `LIMIT` ou `ORDER BY` pour limiter la portée des modifications. La clause `WHERE` est facultative, dans ce cas tous les enregistrements de la table seront affectés. Si `LOW_PRIORITY` est spécifié après `UPDATE`, les modifications auront lieu lorsqu'aucun client ne lira plus la table. Le nombre d'enregistrements affectés est retourné.

`INSERT INTO participants (nom_joueur,score) VALUES ('Manw',1337) ON DUPLICATE KEY UPDATE score=score+1337;`	`ON DUPLICATE KEY UPDATE` procède à une requête `UPDATE` si l'insertion engendre un doublon pour une clé `PRIMARY` ou `UNIQUE`.
`REPLACE INTO animaux (id,espece,nom) VALUES (2,'Elephant','Babar');`	Insère ou remplace des enregistrements dans une table.

Remarque `REPLACE` fonctionne similairement à `INSERT` sauf dans le cas où l'insertion crée un doublon dans une table munie d'un index `UNIQUE` ou `PRIMARY KEY`. Il n'a d'ailleurs de sens qu'avec ces deux types d'index. Dans ce cas, l'ancien enregistrement est effacé avant que le nouveau ne soit inséré.

`DELETE FROM nom_table WHERE condition ORDER BY champ_de_tri LIMIT limite;`	Efface les enregistrements d'une table répondant à une condition, et renvoie le nombre de lignes effacées, éventuellement avec une clause limitative et un ordre de suppression.
`TRUNCATE TABLE nom_table;`	Équivaut en plus rapide à `DELETE FROM nom_table;` car assimilé à une destruction/création de table.
`LOAD DATA INFILE 'contacts.txt' INTO TABLE adresses FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n';`	Insère dans la base de données un fichier texte qui peut avoir été créé grâce à la commande `SELECT ... INTO OUTFILE.`

Remarque L'interprétation du fichier dépend des clauses FIELDS et LINES, définissant la façon dont sont séparés les champs (colonnes) et les lignes du fichier texte.




■ FIELDS peut être spécifié par une (au moins) ou plusieurs instructions :
TERMINATED BY '\t' : colonnes séparées par... (\t = tabulation)
ENCLOSED BY '' : colonnes entourées d'un caractère précis (ex. guillemets)
ESCAPED BY '\\' : caractère d'échappement (ici un antislash simple)
■ LINES est défini par STARTING BY '' (les lignes commencent par...) et TERMINATED BY '\n' (les lignes terminent par...) où \n est une nouvelle ligne ou *line feed* (LF).
Pour ignorer les lignes superflues en début de fichier, IGNORE *nb_de_lignes* LINES.

`SELECT * INTO OUTFILE 'sangria.txt' FIELDS TERMINATED BY ',' FROM boissons;`	Exporte les enregistrements retournés par une instruction **SELECT** vers un fichier texte. (Voir ci-dessus pour la syntaxe des clauses **FIELDS** et **LINES**.)
`SELECT @@version;`	Retourne la version du moteur MySQL utilisé.

Requêtes conditionnelles avec WHERE

L'essentiel étant d'effectuer des requêtes **SELECT**, **REPLACE**, **DELETE** ou **UPDATE** sur les enregistrements d'une table répondant à une condition, **WHERE** se révèle indispensable pour mettre en jeu un ou plusieurs tests, portant sur un ou plusieurs champs. Toutes sortes de fonctions peuvent être utilisées pour comparer les champs entre eux, ou à des constantes, sauf les fonctions d'agrégation (**AVG**, **MIN**, **MAX**, **SUM**...).

Exemple SELECT * FROM *nom_table* **WHERE** *colonne*='*valeur*';

Ces conditions peuvent être multiples grâce aux opérateurs de comparaison.

Opérateurs de comparaison
Booléens : **AND (&&)** : ET logique **OR (||)** : OU logique
Comparateurs : **<** inférieur à **=** égal **<=** inférieur ou égal
> supérieur à **!=** différent de **>=** supérieur ou égal
Parenthèses de priorité :
SELECT * FROM *contacts* **WHERE** *age*>*20* **AND (***code_postal*='*67*' **OR** *age*<=*35*);

LIKE	Teste la validité d'une chaîne de caractères ou d'un nombre par rapport à une expression pouvant contenir deux caractères joker : **%** (aucun ou plusieurs caractères) ; _ (un caractère exactement). **Exemples** SELECT *prenom* FROM *carnet* WHERE *prenom* **LIKE** '*C%*'; **<** *Charles, Céline, Claire* SELECT *mot,definition* FROM *dico* WHERE *mot* **LIKE** *'Fu_ _ _';* **<** *Fusée, Fusil, Futur*
NOT LIKE	Seuls les champs ne correspondant pas à l'expression sont valides.
REGEXP	Teste la conformité du texte à une expression régulière. **Exemple** SELECT *titre* FROM *livres* WHERE *titre* **REGEXP** '*Il était une .+*'; Avec . pour un caractère, [a-z] pour une classe de caractères, * pour 0 ou plusieurs caractères, ^ pour le début de chaîne, $ pour la fin, {n} pour une répétition n-fois. Utiliser **REGEXP BINARY** pour être sensible à la casse.
STRCMP	Compare deux chaînes et renvoie 0 si elles sont identiques, -1 si la première chaîne est plus petite que la seconde, 1 sinon. **STRCMP** est insensible à la casse. **Exemple** SELECT *titre* FROM *livres* WHERE **STRCMP**(*titre*,'*mysql*') >= 0;
IN / NOT IN	Teste l'appartenance d'une valeur à une liste. Depuis MySQL 4.1, **IN()** peut contenir une sous-requête. **Exemple** SELECT *prix* FROM *desserts* WHERE *fruit* **IN**(SELECT *nom* FROM *fruits* WHERE *nom* LIKE '*K%*');
BETWEEN	Teste la présence d'une valeur dans un intervalle, bornes incluses **(***min* **<=** *valeur* **AND** *valeur* **<=** *max***).** **Exemple** SELECT *nom,age* FROM *carnet_adresses* WHERE *age* **BETWEEN** *20* and *35*;
NOT BETWEEN	Seules les valeurs hors de l'intervalle sont prises en compte.
FULLTEXT MATCH... AGAINST	Recherche en texte intégral. N'est possible que sur les colonnes dotées d'un index **FULLTEXT**. Elle est insensible à la casse et la longueur minimale des mots recherchés est de 4 lettres. **Exemple** SELECT * FROM *livres* WHERE **MATCH**(*titre,resume*) **AGAINST** ('*mysql*'); **Pertinence** Quand **MATCH()** est utilisé comme condition de **WHERE** les lignes retournées sont automatiquement organisée par la pertinence. Exprimée sous forme de décimal positif, elle peut être sélectionnée pour opérer un tri personnalisé. *SELECT id*,**MATCH**(*titre,resume*) AGAINST ('*mysql*') AS *pertinence* FROM *livres;* Dans le résultat obtenu figurent tous les enregistrements de la table, assortis d'une colonne *pertinence* dont la valeur est comprise entre 0 (pas de similarité) et 1 (très pertinent).

Déclencheurs (TRIGGER)

Un déclencheur *(trigger)* active une instruction lorsque survient un événement **INSERT**, **UPDATE** ou **DELETE**.

```
CREATE TRIGGER nom_déclencheur BEFORE|AFTER événement ON nom_table FOR EACH ROW requête_à_exécuter;
```

Dans la requête à exécuter, la référence à un champ doit s'accompagner des préfixes **OLD** et **NEW** : **OLD.***nom_champ* et **NEW.***nom_champ* désignent respectivement la ligne *avant* et *après* sa modification (ou avant effacement et après insertion).

Attention Il ne peut exister deux déclencheurs de même type au même moment. CREATE TRIGGER nécessite le privilège SUPER.

Exemple

```
CREATE TRIGGER insertion BEFORE INSERT ON comptes FOR EACH ROW SET @somme = @somme + NEW.montant;
```

Supprimer un déclencheur :

```
DROP TRIGGER nom_table.nom_déclencheur;
```

Groupage et tri

DISTINCT Permet de ne sélectionner qu'une fois chaque valeur différente dans une table (sans doublon).

Exemple

```
SELECT DISTINCT ville FROM annuaire WHERE code_postal LIKE '67%';
```

ORDER BY Ordonne le résultat de la requête en triant sur au moins une colonne.
Par défaut, le tri est croissant (**ASC**). Pour inverser l'ordre, ajouter la clause **DESC** (tri décroissant).

Exemple

```
SELECT nom,prenom FROM ORDER BY age DESC;
```

GROUP BY Groupe les résultats d'une requête selon une ou plusieurs colonnes, en général pour effectuer dessus des calculs (sommes...).

Exemple

```
SELECT joueur,SUM(parties) FROM golf GROUP BY joueur;
SELECT auteur,COUNT(*) FROM commentaires GROUP BY auteur;
```

Total En ajoutant la clause **WITH ROLLUP** après **GROUP BY**, une dernière ligne comprenant le total des valeurs groupées est ajoutée au jeu de résultats.

HAVING Ne peut être utilisé qu'après **GROUP BY**, et, s'il y en a, avant une clause **ORDER BY**. Opère une sous-sélection des résultats à l'aide d'une condition supplémentaire (calcul...). Permet d'utiliser des fonctions non autorisées dans une clause **WHERE**, telles que **SUM**, **AVG**, **MIN**, **MAX**...

Exemple

```
SELECT client,achats,pays FROM clients WHERE pays='France' GROUP BY client,achats HAVING MAX(achats)>1000;
```

LIMIT Limite à ***n*** lignes le résultat retourné, si nécessaire en débutant à partir de la ligne ***m***.

Exemple

```
SELECT * FROM nom_table LIMIT m,n
SELECT nom,prenom FROM contacts LIMIT 5,10;
```

Fonctions mathématiques

La plupart des fonctions sont exploitables dans un **SELECT** et/ou dans une clause **WHERE**.

```
SELECT CONCAT(nom,prenom) FROM adresses WHERE LENGTH(ville)>10;
```

```
SELECT COUNT(*) FROM nom_table;
```
Compte les lignes d'un résultat.

```
SELECT ABS(-1337); < 1337
```
Renvoie la valeur absolue.

```
SELECT CEILING(3.21); < 4
```

```
SELECT FLOOR(3.21); < 3
```
Valeur entière supérieure/inférieure de **x**.

```
SELECT MAX(champ) FROM nom_table;
SELECT MIN(champ) FROM nom_table;
```
Renvoie la valeur maximum/minimum d'un champ.

```
SELECT RAND(); < 0.25508142100189
```
Renvoie un nombre à virgule flottante entre 0 et 1 inclus.

Remarque Peut servir à ordonner un résultat aléatoirement :

```
SELECT * FROM table ORDER BY RAND();
```

```
SELECT ROUND(4.7); < 5
```
Renvoie l'arrondi entier le plus proche.

Nombre de décimales Un deuxième argument *n* peut être précisé pour arrondir la valeur à *n* décimales.

```
SELECT TRUNCATE(4.7); < 4
```
Renvoie la valeur tronquée.

Nombre de décimales Un deuxième argument *n* peut être précisé pour tronquer la valeur à *n* décimales.

Unions et jointures

UNION	Combine les résultats de plusieurs requêtes **SELECT** en un seul résultat. Les colonnes de chaque requête doivent être du même type.

Exemple

```
(SELECT champ1,champ2,... FROM nom_table1) UNION (SELECT champ1,champ2,... FROM nom_table2);
```

Le mot-clé **ALL** renvoie toutes les lignes concernées par chaque **SELECT**. Par défaut, les lignes retournées sont uniques (comme avec **DISTINCT**) pour l'ensemble du résultat.

```
SELECT a,b FROM table1 UNION ALL SELECT c,d FROM table2;
```

Les jointures mettent en relation plusieurs tables grâce à une seule requête (produit cartésien sur lequel on applique des conditions) avec les types de jointure :

INNER JOIN	Sélectionne uniquement les lignes correspondantes qui se trouvent dans les deux tables.
LEFT JOIN	Sélectionne toutes les lignes de la table de gauche et si tel est le cas les entrées correspondantes dans la table de droite. S'il n'y a pas d'entrée correspondante dans la table de droite, elle est remplacée par des valeurs **NULL**.
RIGHT JOIN	Sélectionne toutes les lignes dans la table de droite et les entrées correspondantes dans la table de gauche. S'il n'y en a pas, elle est remplacée par des valeurs **NULL**.
STRAIGHT_JOIN	Identique à **JOIN**, mais la table de gauche est lue avant celle de droite.
NATURAL JOIN	Jointure sur les colonnes qui possèdent le même nom et le même type dans les deux tables.

Ces clauses **JOIN** sont utilisées conjointement avec **ON** ou **USING** : la clause **ON** s'écrit comme une condition utilisée dans une clause **WHERE** ; **USING** nomme les colonnes communes à exploiter pour la jointure.

Equivalence Ces deux instructions sont équivalentes :

```
table1 LEFT JOIN table2 USING (champ1,champ2,champ3)
table1.champ1=table2.champ1 AND table1.champ2=table2.champ2 AND table1.champ3=table2.champ3
```

Exemples

```
SELECT * FROM table1,table2 WHERE table1.id=table2.id;
SELECT * FROM table1 LEFT JOIN table2 ON table1.id=table2.id;
SELECT * FROM table1 LEFT JOIN table2 USING (id);
SELECT * FROM table1 LEFT JOIN table2 USING (champ1,champ2,champ3);
SELECT table1.champ1,table2.champ2 FROM table1 JOIN table2 ON table1.champ3=table2.champ4;
```

Fonctions diverses

SELECT	Renvoie la clé de hachage MD5 (Message-Digest Algorithm) de la chaîne en argument (hexadécimal de 32 caractères).
`MD5('Sheldon');`	❮ *9a70014ca0bb18f7c4e88bf8d9e6de97*
SHA1() ou SHA()	Renvoie la somme SHA1 (Secure Hash Algorithm) de la chaîne passée en argument (hexadécimal de 40 caractères).
`SELECT SHA1('Sheldon Cooper');`	❮ *e6299f9b0450b9de34b8625a6b0c46e014f92afd*
PASSWORD()	Calcule un mot de passe MySQL à partir de la chaîne. Cette fonction est utilisée pour l'encodage des mots de passe stockés dans la table mysql.user.
`SELECT PASSWORD('banane');`	❮ *33b8ada77d684dab*
ENCODE() DECODE()	Ces fonctions encodent et décodent respectivement un texte passé en paramètre à l'aide d'une clé elle aussi spécifiée.
`SELECT ENCODE('C'est un secret','banane');`	❮ *Gc2iF38B4e*
`SELECT DECODE('Gc2iF38B4e','banane');`	❮ *C'est un secret*
FOUND_ROWS()	Renvoie le nombre d'enregistrements trouvés lors de la dernière requête.
LAST_INSERT_ID()	Renvoie le dernier identifiant unique (ex : clé en **AUTO_INCREMENT**) créé par une requête **INSERT**.
VERSION()	Renvoie la version du serveur MySQL utilisé.
`SELECT VERSION();`	❮ *5.1.49-3*
USER()	Renvoie l'identité de l'utilisateur courant.
`SELECT USER();`	❮ *monlogin@localhost*
CONCAT(*chaîne1, chaîne2,...***)**	Concatène des champs ou des chaînes de texte.
`SELECT CONCAT('ki', 'wi');`	❮ *kiwi*
LENGTH(*chaîne***)**	Renvoie la longueur de la chaîne de texte.
`SELECT LENGTH('kiwi');`	❮ *4*

Fonctions texte

INSTR(*chaîne*, *sous-chaîne*)	Renvoie la position de la 1re occurrence de la sous-chaîne. SELECT **INSTR**('J\'*aime les kiwis*','kiwi'); ❮ *12*
LEFT(*chaîne*,*n*)	Renvoie les *n* premiers/derniers caractères de chaîne.
RIGHT(*chaîne*,*n*)	SELECT**LEFT**('J\'*aime les kiwis*', *6*); ❮ *J'aime*
LOWER(*chaîne*)	Met les caractères de chaîne en minuscules/majuscules.
UPPER(*chaîne*)	SELECT **UPPER**('J\'*aime les kiwis*'); ❮ *J'AIME LES KIWIS*
REPLACE(*chaîne*, *chaîne_a*,*chaîne_b*)	Remplace les sous-chaînes ***chaîne_a*** par ***chaîne_b*** dans ***chaîne***. SELECT **REPLACE**('J\'*aime les kiwis*','kiwi','orange'); ❮ *J'aime les oranges*
REVERSE(*chaîne*)	Renvoie une chaîne inversée. SELECT **REVERSE**('*kiwi*'); ❮ *iwik*
SUBSTRING(*chaîne*, *position*,*n*)	Renvoie une sous-chaîne de chaîne, de longueur ***n*** à partir de la position ***position***. SELECT **SUBSTRING**('J\'*aime les kiwis*',*8*,*9*); ❮ *les kiwis*
TRIM(*chaîne*)	Retire les espaces en début et fin de texte.

Fonctions de dates

SELECT **NOW()**; ❮ *2014-01-27 13:37:00* Synonymes **SYSDATE()**, **LOCALTIME()**, **CURRENT_TIMESTAMP()**,**LOCALTIMESTAMP()**	Renvoie la date actuelle au format **'YYYY-MM-DD HH:MM:SS'**.
SELECT **CURDATE()**; ❮ *2014-01-27*	Renvoie la date au format **'YYYY-MM-DD'**.
SELECT **CURTIME()**; ❮ *13:37:00*	Renvoie l'heure au format **'HH:MM:SS'**.
SELECT **UNIX_TIMESTAMP()**; ❮ *1147550227* Remarque Si **UNIX_TIMESTAMP()** est appelée avec un argument date, elle renvoie le timestamp correspondant à cette date.	Renvoie le timestamp UNIX actuel (nombre de secondes écoulées depuis le 1er janvier 1970 à 00:00:00).
SELECT **FROM_UNIXTIME(***1320669158***)**; ❮ *2011-11-07 13:32:28*	Renvoie une date sous la forme **'YYYY-MM-DD HH:MM:SS'** à partir d'un timestamp UNIX.
SELECT **YEAR('***2014-04-27***')**; ❮ *2014* SELECT **DAY('***2014-04-27***')**; ❮ *27* SELECT **MONTH('***2014-04-27***')**; ❮ *04*	Renvoient respectivement l'année, le jour et le mois d'une date.
SELECT **HOUR('***2014-04-27 13:37:00***')**; ❮*13* SELECT **MINUTE('***13:37:00***')**; ❮ *37* SELECT **SECOND('***13:37:00***')**; ❮ *00*	Renvoient respectivement l'heure, les minutes et les secondes d'une date.
SELECT **DAYNAME(NOW())**; ❮ *Sunday* SELECT **MONTHNAME('***2014-04-27***')**; ❮ *April*	Renvoient respectivement le nom du jour et le nom du mois de la date passée en argument.
SELECT **DAYOFWEEK('***2014-04-27***')**; ❮ *1*	Renvoie l'index du jour de la semaine (1 = Dimanche..., 7 = Samedi).
SELECT **DAYOFYEAR('***2014-04-27***')**; ❮ *117*	Renvoie le jour de l'année (1 à 366).
SELECT **WEEK('***2014-04-27***')**; ❮ *17*	Renvoie le numéro de la semaine.
SELECT **DATE('***2014-01-27 13:37:00***')**; ❮ *2014-01-27*	Extrait la partie date d'une expression.
SELECT **TIME('***2014-01-27 13:37:00***')**; ❮ *13:37:00*	Extrait la partie horaire d'une expression.
SELECT **QUARTER('***2014-01-27***')**; ❮ *2*	Renvoie le trimestre de la date (1 à 4).
SELECT **SEC_TO_TIME(***49050***)**; ❮ *13:37:30*	Renvoie l'argument donné en secondes au format **'HH:MM:SS'**.
SELECT **TIME_TO_SEC('***13:37:30***')**; ❮ *49050* secondes.	Convertit en secondes l'argument donné en heures, minutes et
SELECT **ADDTIME('***2014-01-27 00:00:00***', '***13:37:30***')**; ❮ *2014-01-27 13:37:30*	Ajoute une valeur de type **TIME** à une valeur **DATE** ou **DATETIME**.
SELECT **DATEDIFF('***2014-01-27 00:00:00***', '***2014-01-17***')**; ❮ *10*	Renvoie le nombre de jours entre une date de début et une date de fin (types **DATE** ou **DATETIME**).
SELECT **STR_TO_DATE('***2014/01/27 13.37.30***', '%Y/%m/%d %H.%i.%s')**; ❮ *2014-01-27 13:37:30*	Applique un format date à une chaîne de caractères pour renvoyer une valeur **DATETIME**.
SELECT **DATE_FORMAT('***2014-01-27 13:37:30***', '%W %M %Y, %H:%i%:%s')**; ❮ *Monday January 2014, 13:37:30*	Formate une date.
SELECT **DATE_ADD('***2014-01-27 23:59:59***', INTERVAL 1 SECOND)**; ❮ *2014-01-28 00:00:00*	Opérations arithmétiques d'addition et de soustraction sur des dates.
SELECT **DATE_SUB('***2014-01-27***', INTERVAL** *31* **DAY)**; ❮ *2013-12-27*	Remarque Se référer à la documentation MySQL pour la syntaxe des intervalles.



Procédures stockées et fonctions

Une procédure stockée est créée avec **CREATE PROCEDURE** et appelée avec la commande **CALL** ; elle ne renvoie de valeur que via les paramètres de retour (**OUT**). Une fonction peut renvoyer une valeur scalaire, et être appelée depuis une commande SQL, comme toute autre fonction standard.
Une fonction est déclarée avec **CREATE FUNCTION**.
Une variable est déclarée avec la commande *SET@nomvariable=valeur*

Remarque La commande delimiter permet d'utiliser ";" dans la déclaration de la procédure ou de la fonction.

CREATE PROCEDURE

CREATE PROCEDURE *nom_procedure* (*paramètres*) *instructions*

paramètres	Liste des paramètres acceptés par la fonction, suivis de leur type (ex. *nom* **CHAR**(*20*), *age* **TINYINT**). Chaque paramètre est de type **IN** par défaut. Il est possible de spécifier **OUT** ou **INOUT** pour les paramètres de sortie.
instructions	Suite d'instructions SQL, entourées par **BEGIN** et **END**.

Exemple
```
delimiter //
CREATE PROCEDURE proc (OUT poids INT)
BEGIN SELECT SUM(poids_kg) INTO poids FROM commande; END
//delimiter ;
```

Usage **CALL** proc(@*nom_variable*);

Autres éléments de langage :

DECLARE *nom_variable type_variable*;	Déclaration de variable locale
SET *nom_variable* = *expression*;	Affectation de variable
REPEAT ... UNTIL *condition* **END REPEAT;** **WHILE** *condition* **DO ... END WHILE;** **LOOP ... END LOOP;**	Boucles. Pour sortir d'une boucle **LOOP** : **LEAVE**.
IF *condition* **THEN ... ELSE ... END IF;** **CASE** *variable* **WHEN** *valeur* **THEN ... END CASE;**	Conditions
DECLARE *nom_curseur* **CURSOR FOR** *requete_select*;	Déclaration de curseur
OPEN *nom_curseur*;	Ouverture de curseur
FETCH *nom_curseur* **INTO** *variables*;	Lecture de curseur
CLOSE *nom_curseur*;	Fermeture de curseur

CREATE FUNCTION

CREATE FUNCTION *nom_fonction* (*paramètres*) RETURNS *type_retour instructions*

paramètres	Liste des paramètres acceptés par la fonction, suivis de leur type (ex. *nom CHAR(20), age TINYINT)*
type_retour	Type MySQL retourné par la fonction (ex. *CHAR(10), INT)*
instructions	Suite d'instructions SQL, terminées par **RETURN** variable

Exemple
```
delimiter |
CREATE FUNCTION presentation (nom CHAR(20), age TINYINT) RETURNS
CHAR(50)
RETURN CONCAT(nom,' a ',age,' ans');
|delimiter ;
```

Usage `SELECT presentation('Jocelyn',28);` **<** *Jocelyn a 28 ans*

SHOW CREATE PROCEDURE | FUNCTION

SHOW CREATE PROCEDURE *nom_procedure*; **SHOW CREATE FUNCTION** *nom_fonction*;	Renvoie l'instruction permettant de recréer la procédure ou la fonction.
DROP PROCEDURE *nom_procedure*; **DROP FUNCTION** *nom_fonction*;	Supprime une procédure ou une fonction.

Transactions

Une transaction n'est effective que sur une table de type InnoDB ou BDB (mode autocommit désactivé : `SET AUTOCOMMIT=0;`). Elle permet de revenir si nécessaire (en cas d'erreur) à l'état précédent, avant le début de la transaction. Elle débute par **START TRANSACTION** et se termine par **COMMIT**.
ROLLBACK restaure l'état initial de la table avant la fin de la transaction sans appliquer les modifications.

```
START TRANSACTION;
UPDATE nom_table SET modifications;
UPDATE nom_table SET autres_modifications;
COMMIT;
```

```
START TRANSACTION;
UPDATE nom_table SET ;
ROLLBACK;
```

Administration et performances

`SHOW GRANTS FOR` *login*;	Affiche les droits attribués à l'utilisateur login sous forme de requête SQL.
`SHOW GRANTS FOR CURRENT_USER`;	

`SHOW INDEX FROM` *nom_table*;	Affiche les index de la table.

Les champs suivants sont retournés :

`Table`	Nom de la table.
`Non_unique`	1 si les doublons sont autorisés, 0 sinon.
`Key_name`	Nom de l'index
`Seq_in_index`	Position de la colonne dans l'index
`Column_name`	Nom de la colonne indexée
`Collation`	Mode de tri de l'index ('A' tri ascendant, `NULL` pas de tri)
`Cardinality`	Nombre de valeurs uniques de l'index
`Sub_part`	Nombre de caractères si la colonne est partiellement indexée, `NULL` sinon.
`Packed`	Mode de compactage de la clé
`Null`	YES si la colonne peut contenir `NULL`
`Index_type`	Type d'index (`BTREE`,`FULLTEXT`,`HASH`,`RTREE`)
`Comment`	Informations complémentaires

`SHOW STATUS;`	Informe sur l'état du serveur :
`SHOW STATUS LIKE '`*Table*`%';`	variables d'exécution, liste des processus
`SHOW VARIABLES;`	en cours d'exécution, avec la requête
`SHOW VARIABLES LIKE '`*max*`%';`	exécutée correspondante, l'utilisateur
`SHOW PROCESSLIST;`	et le numéro de processus, etc.

`KILL` *numero_processus*;	Termine un processus grâce à la commande `KILL` suivie du numéro de processus concerné.

Remarque Seuls les droits d'administrateur (SUPER) permettent de voir la liste de tous les processus et de les terminer; à défaut seuls les processus de l'utilisateur courant sont accessibles.

EXPLAIN

Performances Cette commande est essentielle à l'optimisation de tables.

En faisant précéder une requête `SELECT` avec le mot-clé `EXPLAIN`, MySQL renvoie un enregistrement décrivant comment le moteur traitera la requête :

`id`	Identifiant de requête
`select_type`	Type de requête `SELECT`
`table`	Table utilisée pour la requête
`type`	Type de jointure
`possible_keys`	Index utilisables pour effectuer la requête
`key`	Index réellement utilisé
`key_len`	Taille de la clé d'index utilisée (plus la valeur est petite plus l'index est rapide)
`ref`	Références utilisées avec la clé key pour effectuer la requête
`rows`	Estimation du nombre de lignes à parcourir dans la table pour effectuer la requête
`extra`	Informations complémentaires sur la méthode appliquée.

Réparer et restaurer

`ANALYZE TABLE` *nom_table*;	Analyse et stocke la clé de distribution de la table qui permet de décider dans quel ordre les tables doivent être rassemblées lors des jointures qui ne s'effectuent pas sur une constante.
`OPTIMIZE TABLE` *nom_table*;	Défragmente une table. Optimise les performances suite à un grand nombre d'insertions et effacements,
`CHECK TABLE` *nom_table*;	Vérifie l'intégrité d'une table.
`REPAIR TABLE` *nom_table*;	Tente la réparation d'une table corrompue.
`LOCK TABLES` *table1*,*table2* `READ`;	Verrouille une ou plusieurs tables en lecture.
`LOCK TABLES` *table1*,*table2* `WRITE`;	Verrouille une ou plusieurs tables en écriture.

Remarque Seule la connexion courante peut lire et écrire sur cette table. Les autres connexions sont bloquées en écriture.

`UNLOCK TABLES`;	Déverrouille les tables précédemment spécifiées.
`BACKUP TABLE` *nom_table* `TO '`*/chemin/sur/le/disque*`';`	Sauvegarde les fichiers de définition et de données de la table vers le chemin spécifié, afin de la restaurer ultérieurement. Seules les tables MyISAM sont supportées.
`RESTORE TABLE` *nom_table* `FROM '`*/chemin/sur/le/disque*`';`	Restaure une table à partir d'une sauvegarde effectuée avec `BACKUP TABLE`, stockée sur le disque dans le dossier *chemin*.

Vues

`CREATE VIEW` *nom_de_la_vue* `AS` *requete_select*;	Crée une nouvelle vue d'après une requête `SELECT` qui fournit la définition de la vue.
`CREATE VIEW` *memento* `AS` SELECT *titre*,*prix* FROM *livres* WHERE type='*memento*';	
`SHOW CREATE VIEW` *memento*	Affiche la commande qui a créé la vue.
`<` SELECT *titre*,*description*,*prix* FROM *livres* WHERE type='*memento*'	
`ALTER VIEW` *nom_de_la_vue* `AS` *nouvelle_requete_select*;	Modifie une vue existante en lui assignant une nouvelle définition.
`ALTER VIEW` *memento* `AS` SELECT *titre*,*description* FROM *livres* WHERE *prix*='5';	
`DROP VIEW` *nom_de_la_vue1*`,` *nom_de_la_vue2*`,...;`	Supprime une ou plusieurs vues (séparées par des virgules).

Sécurité et injections SQL

En PHP il faut « échapper » les variables provenant des utilisateurs avec **`mysqli_real_escape_string`** qui remplace les caractères pouvant être pris pour des caractères spéciaux SQL. La connexion doit d'abord être établie pour pouvoir l'utiliser :

```
if(isset($_POST['prenom'])){
   $prenom = mysqli_real_escape_string($_POST['prenom']);
mysqli_query("INSERT INTO inscrits (prenom) VALUES ('$prenom')");
```

Les *prepared statements* permettent dans différents langages de s'affranchir de cette précaution, avec les commandes **`PREPARE`** et **`EXECUTE`**.

Précautions à prendre ! 1. Vérifier, valider, échapper les données entrées par l'utilisateur. **2.** Bloquer toute donnée pouvant contenir un mot-clé SQL interdit. **3.** Ne définir que des autorisations d'accès minimales par utilisateur MySQL (à l'aide de GRANT). **4.** Utiliser des procédures stockées et des requêtes paramétrées.

Variables serveur

En PHP, les variables serveur sont définies en priorité par les commandes passées en argument au démarrage du serveur **`mysqld`** et dans les fichiers de configuration : **`/etc/mysql/my.cnf`** ou **`/etc/my.cnf`** sous Linux et **`C:\Windows\my.ini OU C:\my.cnf`** sous Windows. Pour en voir la liste : **`SHOW VARIABLES;`**

Variables courantes

`port` = *3306*	Port utilisé par le serveur. Par défaut : 3306
`socket` = */var/run/mysqld/mysqld.sock*	Socket à utiliser
`basedir` = */usr*	Dossier d'installation de MySQL
`datadir` = */var/lib/mysql*	Dossier de données de MySQL
`tmpdir` = */tmp*	Chemin utilisé pour le stockage des fichiers temporaires
`language` = */usr/share/mysql/french*	Langue utilisée par le serveur pour afficher les messages
`key_buffer_size` = *16M*	Taille du buffer utilisé pour le cache des blocs d'index
`log` = */var/log/mysql/mysql.log*	Log de toutes les requêtes
`log-bin` = */var/log/mysql/mysql-bin.log*	Log binaire
`log-slow-queries` = */var/log/mysql/mysql-slow.log*	Log des requêtes les plus lentes, dont la durée est spécifiée par la variable long_query_time, en secondes
`max_connections` = *128*	Nombre maximal de clients MySQL simultanés
`query_cache_limit` = *1M*	Ne met pas en cache les résultats plus grands que la valeur spécifiée
`query_cache_size` = *16777216*	Mémoire allouée pour la mise en cache des requêtes précédentes

Remarque Certaines de ces variables sont modifiables dynamiquement durant l'exécution grâce à l'instruction **`SET GLOBAL`** pour l'ensemble du serveur, ou **`SET SESSION`** pour la session courante uniquement : **`SET GLOBAL`** `query_cache_limit=`*1048576*;

Chez le même éditeur…

HTML 5, R. Rimelé, 2013
MySQL 5 – Audit et optimisation, Borghino, 2010
Mémento PHP 5 et SQL, C. Pierre de Geyer et al., 2009
Mémento Unix/Linux, 4e éd., I. Hurbain, E. Dreyfus, 2011
Mémento CSS 3, R. Goetter, 2013
Sites web, les bonnes pratiques, E. Sloïm, 2010
Mémento HTML 5, R. Rimelé, 2014

Code éditeur : G14039
ISBN : 978-2-212-14039-2
Conception : Nord Compo