Support de cours

Introduction à SQL et MySQL

Sommaire

SQL & MySQL

Le langage SQL (Structured Query Language) est un langage de requête utilisé pour interroger des bases de données exploitant le modèle relationnel.

SQL fait l'objet d'une norme ANSI. Cependant, la quasi-totalité des serveurs de bases de données proposent des extensions qui rendent les programmes difficilement portables.

MySQL (dans sa version 3) implémente un sous-ensemble de la norme ANSI SQL92.

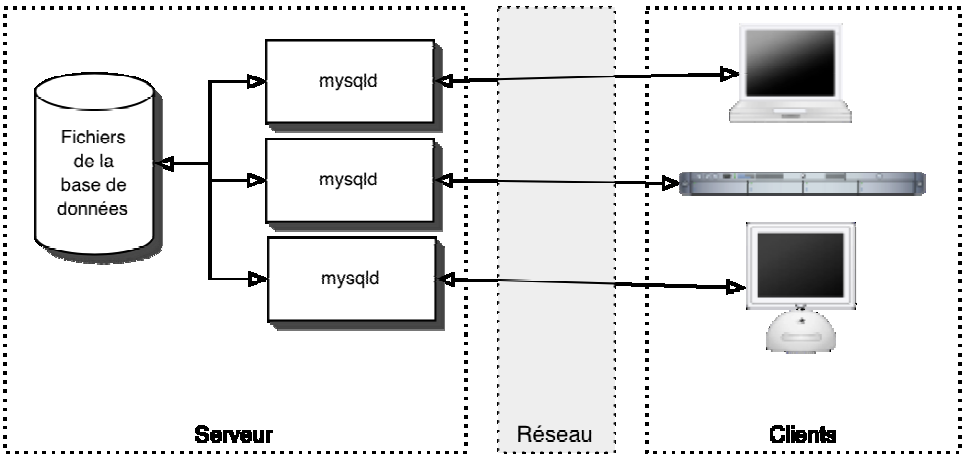
Les points forts de MySQL sont :
- implémentation libre et populaire ;
- facile à mettre en œuvre ;
- rapide à apprendre ;
- support multi-plateforme ;
- fiable et rapide.

Ses principaux points faibles sont :
- ne possède pas de mécanisme transactionnel dans sa version 3 ;
- n'implèmente pas les références d'intégrité relationnelles ;
- absence de procédures stockées et triggers.

Architecture

MySQL est basé sur une architecture client/serveur.

C'est-à-dire que les clients doivent s'adresser au serveur qui gère, contrôle et arbitre les accès aux données.



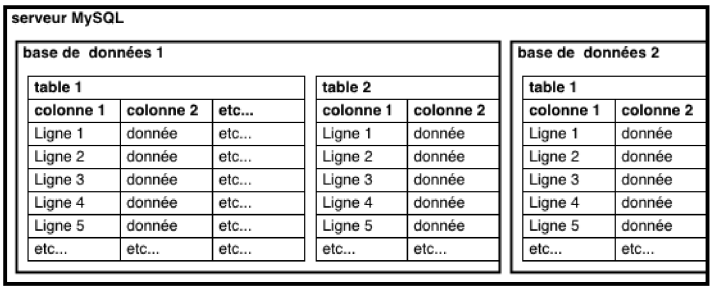
Objets

Un **serveur** MySQL gère une ou plusieurs **base de données**.
Chaque base de données contient différents types d'objets (tables, index, fonctions).
L'objet le plus représenté d'une base de données est la **table**.
Chaque table (appelées encore « relation ») est caractérisée par une ou plusieurs **colonnes** (ou « attributs »).

Le langage qui permet de gérer ces objets est appelé « *Langage de Description des Données* » (LDD).

Les données sont stockées dans les tables sous forme de **lignes** (ou « tuples »).

Le langage qui permet de manipuler les données est appelé « *Langage de Manipulation des Données* » (LMD).



Clés primaires et étrangères

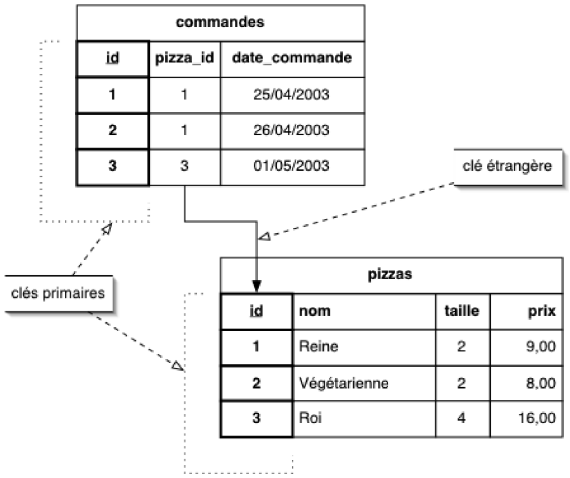
Une table contient généralement une **clé primaire**.
Une clé primaire est constituée d'une ou plusieurs colonnes.

Les valeurs des colonnes qui constituent la clé primaire d'une table sont uniques pour toutes les lignes de la table. La clé primaire d'une table permet donc de faire référence de manière univoque à chaque ligne de la table.

Par exemple, les tables `pizzas` et `commandes` possèdent toutes deux une clé primaire qui est constituée de leur colonne `id` respective.

La colonne `pizza_id` de la table `commandes` fait référence à la colonne `id` (donc à la clé primaire) de la table `pizzas`.

Ce type de référence est appelée « **clé étrangère** ». MySQL n'implémente pas les clés étrangères sous forme de contrainte.



Clés uniques

En complément à la clé primaire, une ou plusieurs clés uniques (appelées également « clés secondaires » ou « clés alternatives ») peuvent être associées à une table.

Les clés uniques sont identiques aux clés primaires à la différence prés que les colonnes qui les constituent peuvent contenir une valeur NULL.
La plupart du temps, toute table devrait posséder au moins une clé primaire.

Par exemple, si la clé primaire de la table `pizzas` est constituée de la colonne `id` et s'il existe une clé unique constituée de la seule colonne `nom`, le schéma ci-contre propose trois nouvelles lignes à l'insertion dont une seule pourra être ajoutée à la table.

pizzas

id	***nom***	**taille**	**prix**	
1	Reine	2	9,00	
2	Végétarienne	2	8,00	
3	Roi	4	16,00	
3	Calzone	4	13,00	insertion interdite
4	Roi	2	11,00	insertion interdite
4	Calzone	4	14,00	insertion ok

Interfaces à MySQL

La plupart du temps l'accès aux bases de données se fait via un autre langage de programmation (C, Perl, PHP, etc.).

Cependant, il est régulièrment nécessaire d'exécuter des requêtes sur la base de données, ne serait-ce que pour construire les objets avec le LDD.

L'interface qui est traditionnellement utilisée est l'utilitaire `mysql`. Il s'agit d'un programme en ligne de commande qui permet d'exécuter des requêtes SQL et d'en visualiser les éventuels résultats. Un exemple est donné page suivante.

Cependant, grâce à la grande popularité de MySQL, d'autres interfaces plus conviviales ont vu le jour. Les plus utilisées sont :

- phpMyAdmin est, comme son nom l'indique, une interface Web écrite en PHP ;
- Webmin, il existe un module d'administration de MySQL pour Webmin ;
- WinMySQLAdmin est une interface pour les systèmes Windows ;
- MySQLCC (« MySQL Control Center », `http://www.mysql.com/products/mysqlcc`).

L'interface la plus répandue est phpMyAdmin.

Interface à MySQL

Cet exemple montre comment établir une connexion au serveur MySQL à partir d'un shell Unix.

Cette connexion est ouverte avec le nom d'utilisateur « pizzeria » (`-u pizzeria`), un mot de passe devra être entré (`-p`) et la base de données à laquelle est demandée la connexion est « pizzas ».

Une fois la connexion établie, un prompt (`mysql>`) attends les requêtes SQL.

Une requête simple de sélection est effectuée sur la table « pizzas ».
Le résultat de cette requête (qui est l'ensemble des lignes de la table) est affiché.

```
seb@puck:~$ mysql -u pizzeria -p pizzas
Enter password:
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 57 to server version: 3.23.49-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> select * from pizzas;
+----+--------------+--------+------+
| id | nom          | taille | prix |
+----+--------------+--------+------+
|  1 | Reine        |      2 |    9 |
|  2 | Végétarienne |      2 |    8 |
|  3 | Roi          |      4 |   16 |
+----+--------------+--------+------+
3 rows in set (0.01 sec)

mysql> quit
Bye
seb@puck:~$
```

phpMyAdmin, installation

phpMyAdmin est disponible sur ce site : `http://www.phpmyadmin.net`

Pour installer phpMyAdmin, il est nécessaire d'avoir au préalable un serveur Web fonctionnel (par exemple Apache) incluant PHP et le support MySQL pour PHP.

Voici les étapes d'installation de phpMyAdmin :

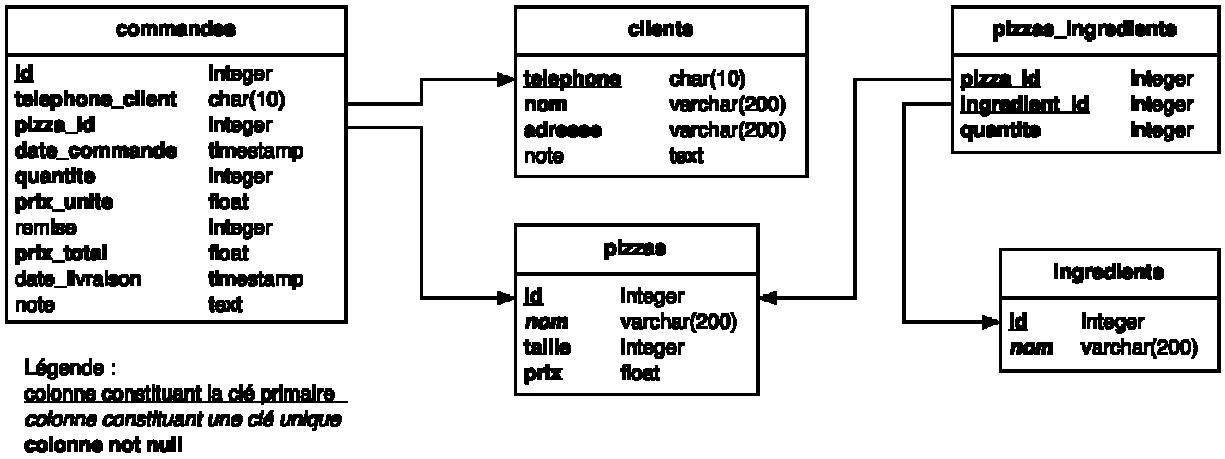
```
puck:~# cd /var/www
puck:/var/www# tar xzf /root/phpMyAdmin-2.5.1-rc1-php.tar.gz
puck:/var/www# mv phpMyAdmin-2.5.1-rc1 phpMyAdmin
puck:/var/www# cd phpMyAdmin
puck:/var/www/phpMyAdmin# vi config.inc.php
[ Modifier les lignes suivantes :                                    ]
[ $cfg['PmaAbsoluteUri']          = 'http://puck.anet.fr/phpMyAdmin';  ]
[ $cfg['Servers'][$i]['auth_type'] = 'cookie';                         ]
```

Remplacer la chaîne `puck.anet.fr` par le nom du serveur MySQL.
À partir d'un navigateur Web, il est alors possible d'administrer le serveur MySQL en entrant l'URL suivante : *`http://puck.anet.fr/phpMyAdmin`*

Base de données pizzas

Tout au long de ce support de cours les exemples se réfèreront à une base de données fictive. Cette bases modélise (très simplement) une activité de fabrication et de livraison de pizzas.

Le Modèle Physique de Données (MPD) est un diagramme qui permet de présenter les tables avec leurs colonnes ainsi que les références entre elles.



LMD, verbes

Le langage de manipulation des données (LMD) est constitué de quatre verbes :

```
SELECT INSERT UPDATE DELETE
```

Chacun de ces verbes possède une ou plusieurs clauses.

Lorsqu'elles sont entrées dans l'utilitaire `mysql`, les commandes SQL doivent se terminer par un point-virgule. Dans d'autres interfaces le point-virgule final est généralement accepté mais pas souvent nécessaire.

Par exemple :

```
select * from pizzas where nom='Roi';
```

La clause `from` permet de spécifier le nom de la table qui est interrogée, la clause `where` permet de préciser un critère de sélection.

Les chaînes de caractères sont entourées par des apostrophes simples. Si une apostrophe simple est présente dans la chaîne, elle doit être doublée. Par exemple :

```
'Il faut s''asseoir pour réfléchir !'
```

LMD, `select`

Le verbe `SELECT` permet de faire des requêtes sur une ou plusieurs tables.
Il ne modifie jamais les données.

Une forme simple de `select` est :

```
select colonnes from tables where condition order by colonnes[asc/desc]
```

Par exemple, la requête suivante affiche le nom et le prix des pizzas pour deux personnes. Le résultat est trié par prix :

```
select nom, prix from pizzas where taille=2 order by prix;
```

Pour sélectionner toutes les colonnes d'une table, on utilise le caractère « * ». Par exemple :

```
select * from ingredients order by nom desc;
```

Il est également possible de réaliser des calculs.
Combien coûtent trois pizzas reines ?

```
select prix*3 from pizzas where nom='Reine';
```

LMD, `select`

Les opérateurs de comparaison sont les suivants :

- arithmétiques : `= < > <= >= !=`
- plage de valeur : `between` *`v1`* `and` *`v2`*
- appartenance : `in (`*`v1`*`, `*`v2`*`, etc.)`
- nullité : `is null`, `is not null`
- comparaison de chaîne : `like '`*`modèle`*`'` (caractères joker : « `%` » et « _ »)

Exemples :

```
select * from commandes where quantite >= 2;
select nom, prix from pizzas where prix between 5 and 10;
select * from ingredients where id in (1, 2, 4);
select nom, note from clients where note is not null;
select telephone, nom from clients where telephone like '01%' order by nom;
```

Opérateur booléens : `and` et `or`.

Exemple :

```
select * from pizzas where taille = 4 and prix < 10;
```

LMD, `select`

Le verbe select dispose d'une clause qui permet de limiter le nombres d'enregistrements retournés : `limit`.

Elle s'utilise ainsi :

```
select colonnes from table where condition limit [a,] b
```

Où *a* est l'index (à partir de 0) de la première ligne à afficher (0 si non précisé) et *b* est le nombre maximal de lignes. Si *b* est « `-1` », toutes les lignes restantes sont retournées.

Par exemple, pour lister les lignes de la table `commandes` trois par trois, on utiliserait successivement :

```
select * from commandes limit 3;
select * from commandes limit 3, 3;
select * from commandes limit 6, 3;
```

etc...

Cette clause est particulièrement utile lorsque MySQL est utilisé pour afficher le contenu d'une table qui possède beaucoup de lignes (par exemple via une application Web).

LMD, `select`

Les fonctions de groupes permettent de répondre à des questions du type « quelle est le nombre de clients de la pizzéria ? ». Une telle question s'écrirait ainsi :

```
select count(*) from clients;
```

Les fonctions de groupes sont : `count sum max min avg`.
Elles sont également appelées « fonctions agrégeantes » lorsqu'elles sont utilisées avec la clause `group by`.

Exemple :

```
select telephone_client, sum(prix_total)
    from commandes group by telephone_client;
```

Il peut être pratique d'utiliser les alias de colonnes. Par exemple, pour établir le palmarès des trois pizzas les plus vendues :

```
select pizza_id as pizza_no, sum(quantite) as total
    from commandes group by pizza_id order by total desc limit 3;
```

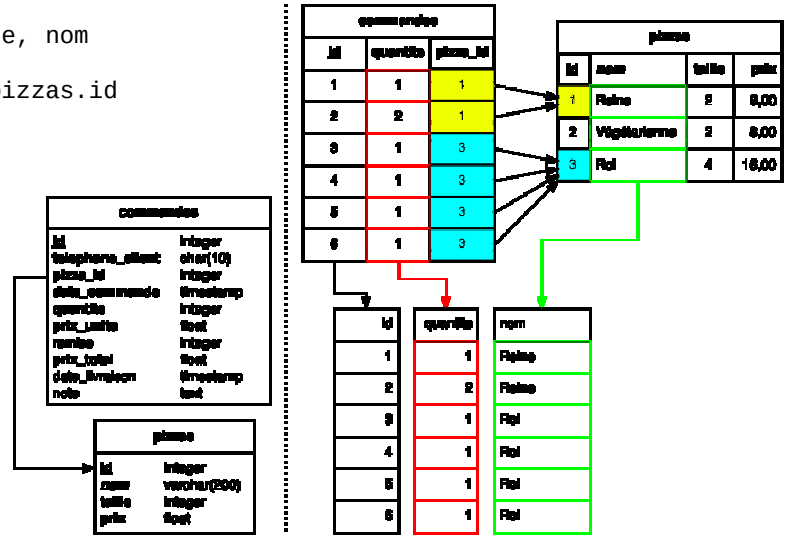
LMD, jointures

Pour obtenir la liste et la quantité des pizzas commandées, il est nécessaire d'extraire des données provenant des tables commandes (quantités commandées) et pizzas (nom des pizzas). L'identifiant de la pizza (donnée présente dans les deux tables) formera le pivot de cette requête. Cette technique est appelée jointure.

```
select commandes.id, quantite, nom
from commandes, pizzas
where commandes.pizza_id = pizzas.id
order by 1;
```

Le schéma ci-contre montre côte à côte un morceau du MPD de la base de données, le contenu des tables commandes et pizzas et le résultat de la requête ci-dessus.

Il met en relief le fait que les jointures ne sont que l'incarnation des liens relationnels qui existent entre les tables de la base de données.



LMD, jointures

Il existe deux formes pour écriture les jointures :

```
select commandes.id, commandes.quantite, pizzas.nom
from commandes, pizzas
where commandes.pizza_id = pizzas.id
order by 1;

select commandes.id, commandes.quantite, pizzas.nom
from commandes
inner join pizzas on commandes.pizza_id = pizzas.id
order by 1;
```

Ces deux requêtes fournissent le même résultat.

LMD, `jointures`

Si nous reprenons notre exemple de palmarès des pizzas vendues, il serait bien pratique d'afficher en première colonne le nom de la pizza plutôt que son identifiant.

Ceci est possible en utilisant une jointure.

```
select nom, sum(quantite) as total
from commandes, pizzas
where commandes.pizza_id = pizzas.id
group by pizza_id
order by total desc
limit 3;
```

LMD, jointures

Une jointure peut se faire avec plus de deux tables.

Ainsi, si nous souhaitons obtenir la liste et la quantité des ingrédients pour chaque pizza (afin de l'afficher en cuisine) :

```
select pizzas.nom as pizza, ingredients.nom as ingredient, pizzas_ingredients.quantite
from pizzas, ingredients, pizzas_ingredients
where pizzas_ingredients.pizza_id = pizzas.id
  and pizzas_ingredients.ingredient_id = ingredients.id
order by 1;
```

Le nom de chaque colonne doit être préfixé par le nom de la table dont il est issu car il y aurait sinon ambiguïté. Il est également possible d'utiliser des alias pour les tables :

```
select piz.nom as pizza, ing.nom as ingredient, pzi.quantite
from pizzas piz, ingredients ing, pizzas_ingredients pzi
where pzi.pizza_id = piz.id
  and pzi.ingredient_id = ing.id
order by 1;
```

LMD, `insert`

Le verbe `insert` permet d'ajouter des lignes à une table.

Il s'utilise de deux manières :

```
insert into table (colonnes) values (valeurs)
insert into table (colonnes) select colonnes from tables
```

La liste des colonnes entre parenthèses est optionelle. Si elle est omise la liste des valeurs doit fournir une valeur pour toute les colonnes de la table dans l'ordre dans lequel elle ont été spécifiées lors de la création de la table.
La seconde forme permet d'insérer dans la table de destination le résultat d'une requête.

Par exemple :

```
insert into ingredients (id, nom) values (8, 'Poivron');
```

est équivalent à :

```
insert into ingredients values (8, 'Poivron');
```

si les colonnes `id` et `nom` sont ordonnées de cette manière dans la table `ingredients`.
Pour le vérifier :

```
desc ingredients;
```

LMD, `insert`

Il n'est pas nécessaire de fournir une valeur pour les colonnes dites « nullable » (c'est-à-dire créée avec l'option `not null`).

Il est possible d'effectuer une insertion à partir du résultat d'une requête tout en fournissant certaines valeurs « externes ». Par exemple, pour créer une commande de deux pizzas « Roi » pour le client Alex Feinberg :

```
insert into commandes
 (telephone_client, pizza_id, date_commande, quantite, prix_unite, prix_total)
select
 clients.telephone, pizzas.id, now(), 2, pizzas.prix, pizzas.prix*2
from pizzas, clients
where pizzas.nom  = 'Roi'
  and clients.nom = 'Alex Feinberg';
```

LMD, `update`

Le verbe `update` permet de modifier des lignes déjà présentes dans une table.

Sa syntaxe est la suivante :

```
update table set colonne=valeur, colonne=valeur where condition
```

La clause where est optionelle. Si elle est omise, les modifications sont appliquées à la totalité des lignes de la table.

Par exemple, nous livrons la commande de d'Alex Feinberg à 20h28 :

```
update commandes set date_livraison = '2003-05-14 20:28' where id = 7;
```

Il est possible d'introduire une formule dans la requête de mis-à-jour.

Par exemple, nous décidons d'augmenter de 10% le prix de toutes nos pizzas pour deux personnes :

```
update pizzas set prix = prix*1.1 where taille = 2;
```

LMD, `delete`

Le verbe `delete` est utilisé pour supprimer des lignes d'une table.

Sa syntaxe est la suivante :

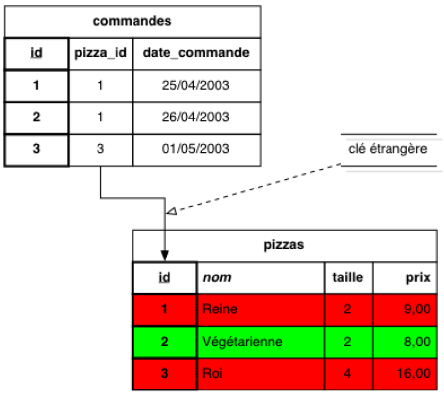
```
delete from table where condition
```

La clause `where` est optionelle. Toutes les lignes sont supprimées si elle est omise.

Dans une base de données relationnelle, il n'est pas possible de supprimer des lignes d'une table si des lignes d'une autre table font référence à l'une des valeurs de ces lignes (clé étrangère).

Cependant MySQL n'implémente pas ce genre de contraint d'intégrité. Il conviendra donc de vérifier au préalable que de telles références n'existe pas avant d'effectuer la suppression.

Dans le schéma ci-contre, seule la ligne surlignée en vert peut être supprimée sans risque de la table `pizzas`.



LDD, verbes

Le langage de description de données (LDD) est constitué de trois verbes :

```
CREATE DROP ALTER
```

Ce langage permet de décrire les objets qui contiendront les données. Il ne sert pas à manipuler ces données.

En général il n'est utilisé que lors de la création du schéma de la base de données au moment de l'installation du logiciel ou bien lors de la modification de ce schéma si des mises-à-jour sont effectuées.

LDD, `create table`

Le verbe `create table` est utilisé pour créer une table avec l'ensemble de ses colonnes.

Sa syntaxe est :

```
create table table
  ( colonne1 type [not null]
  , colonne2 type [not null]
  , ...
  , primary key (colonnes)
[ , unique (colonnes) ]
  , ... )
```

Les colonnes qui constituent la clé primaire doivent toutes être `not null`.

LDD, `create table`

Les types de données les plus utilisés sont :

varchar(n)	chaîne de caractères de taille variable
char(n)	chaîne de caractères de taille fixe
integer	entier
float	nombre à virgule flottante
date	date
time	heure
timestamp	date et heure (positionné à la date et l'heure actuelle si mis à NULL)
datetime	date et heure
text	textes longs
blob	valeur binaire longue
enum(liste)	énumération

À noter :

- les floats conduisent facilement à des erreurs d'arrondi, il est conseillé d'utiliser des entiers pour stocker des valeurs numéraires ;
- il existe plusieurs modificateurs et types moins utilisés, consulter la documentation MySQL pour plus d'information.

LDD, `create table`

Pour gérer les livraisons, nous avons plusieurs livreurs. Nous allons donc ajouter une table nous permettant d'indiquer qui livre chaque commandes. Un livreur est caractérisé par son identifiant unique, son prénom et sa date d'embauche. De plus une colonne permet d'entrer un commentaire.

```
create table livreurs
( id            integer     not null
, prenom        varchar(50) not null
, date_embauche date
, commentaire   text
, primary key (id)
, unique (prenom) );
```

Ou bien :

```
create table livreurs
( id            integer      not null primary key
, prenom        varchar(200) not null unique
, date_embauche date
, commentaire   text );
```

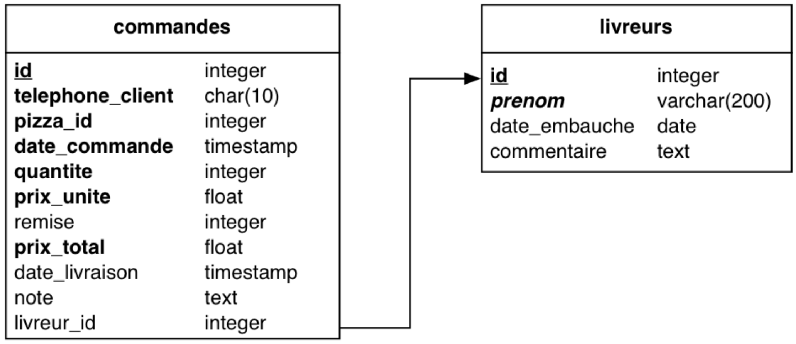
livreurs	
id	integer
prenom	varchar(200)
date_embauche	date
commentaire	text

LDD, `alter table`

La commande `alter table` permet de modifier certaines caractéristiques d'une table ou de l'une de ses colonnes.

Elle est fréquemment utilisée pour ajouter une colonne à un table.
Par exemple, il nous est maintenant nécessaire d'ajouter la colonne `livreur_id` à la table `commandes`. Cette colonne contiendra l'identifiant du livreur ou la valeur NULL si le client est venu chercher la pizza sur place.

```
alter table commandes add column livreur_id integer;
```



LDD, `alter table`

La commande `alter table` permet également :

- d'ajouter une clé unique
  ```
  alter table table add unique (colonnes)
  ```
- supprimer une colonne
  ```
  alter table table drop column colonne
  ```
- changer le nom et le type d'une colonne
  ```
  alter table table change column colonne nouveau_nom nouveau_type
  ```
- renommer une table
  ```
  alter table table rename as nouveau_nom
  ```

La commande `drop table` supprime une table :

```
drop table table
```

Bases de données

Les commandes `create database` et `drop database` permettent de créer ou de supprimer une base de données.

Leur syntaxe est simple. Attention lors de l'utilisation de `drop database` !

```
create database nombase
drop database nombase
```

L'utilisation de ces commandes n'est autorisée que si les droits de l'utilisateur connecté le permettent.

Dans l'utilitaire mysql, pour changer de base de données, utiliser la commande `use` ainsi :

```
use nombase
```

Par exemple, un script de création d'une base de données commence souvent ainsi :

```
create database pizzas;
use pizza;
create table...
```

Droits d'accès

Le système de gestion des droits d'accès de MySQL est basé sur cinq tables de la base de données `mysql` (créée lors de l'installation de MySQL) :

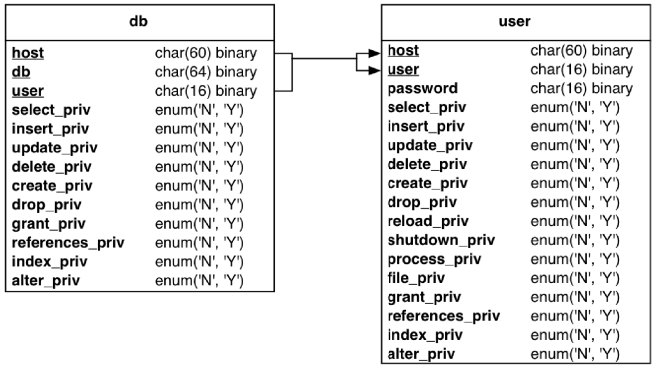
```
user db host tables_priv column_priv
```

Dans le cadre de ce support de cours, nous nous limiterons à l'étude des deux premières.

Un utilisateur est identifié par le couple : user, host. La syntaxe utilisée est « `user@host` ».

Des privilèges sont associés à chaque utilisateur soit au niveau global, dans la table `user`, soit au niveau d'une base de données, dans la table `db`.

Note : Pour rappel, la contrainte d'intégrité référentielle montrée dans le MPD ci-contre n'est pas implémentée dans MySQL. Elle n'est représentée que dans un soucis didactique.



Droits d'accès

La table `user` contient la liste des utilisateurs qui ont accès au serveur MySQL.
Leur mot de passe est stocké dans la colonne `password`.
Cette table contient également deux types de privilèges :

- les privilèges associés au serveur MySQL
  ```
  create(database) drop(database) reload shutdown process file
  ```
- les privilèges associés aux tables de toutes les bases de données
  ```
  select insert update delete create(table,index) drop(table)
  grant references index alter
  ```

Ainsi, un utilisateur qui possède le privilège `create` dans la table `user` est autorisé à créer des bases de données et des tables dans toutes les bases de données.
Il lui est également permis de créer des index dans toutes les bases de données mais uniquement lors de la création d'une table avec la commande `create table` (requis pour la création des clés primaires et uniques).

En général, il existe un seul utilisateur MySQL qui possède tous les droits au niveau global. Il est considéré comme l'administrateur du serveur MySQL. Bien souvent, il s'agit de « `root@localhost` ».

Droits d'accès

La table `db` précise les privilèges des utilisateurs pour chaque base de données.

Les privilèges contenus dans cette table sont pris en compte uniquement si le même privilège est « `N` » dans la table `user` pour le même couple user@host.

L'exemple ci-contre montre une configuration classique où l'utilisateur root@localhost possède tous les privilèges (il est considéré comme administrateur) et l'utilisateur pizzeria@localhost ne possède aucun privilège au niveau global et presque tous les privilèges sur une seule base de données.

user

host	localhost	localhost
user	root	pizzeria
password	378b243...	jh7i875o...
select_priv	Y	N
insert_priv	Y	N
update_priv	Y	N
delete_priv	Y	N
create_priv	Y	N
drop_priv	Y	N
reload_priv	Y	N
shutdown_priv	Y	N
process_priv	Y	N
file_priv	Y	N
grant_priv	Y	N
references_priv	Y	N
index_priv	Y	N
alter_priv	Y	N

db

host	localhost
db	pizzas
user	pizzeria
select_priv	Y
insert_priv	Y
update_priv	Y
delete_priv	Y
create_priv	Y
drop_priv	Y
grant_priv	N
references_priv	Y
index_priv	Y
alter_priv	Y

Droits d'accès

Il existe deux méthodes pour gérer les utilisateurs et les privilèges qui y sont associés :
- modifier les tables `user`, `db`, etc. en utilisant le LMD ;
- utiliser les commandes SQL `grant`, `revoke` et `set password`.

La seconde méthode est désormais préférée à la première.

Si l'une des tables `user`, `db`, `host`, `tables_priv` ou `column_priv` est modifiée, il est nécessaire d'utiliser l'instruction `flush privileges` afin que MySQL prenne en compte les changements.

Par exemple :

```
update user set password=password('Z') where user='root' and host='localhost';
flush privileges;
```

Ceci n'est pas nécessaire lorsque les commandes `grant`, `revoke` et `set password` sont utilisées. Cette commande est équivalente aux deux commandes de l'exemple précédent :

```
set password for root@localhost = password('Z');
```

Droits d'accès, `grant`

La commande `grant` permet d'allouer des privilèges à un utilisateur. Celui-ci est créé s'il n'existe pas dans la table `user`. Sa syntaxe est :

```
grant privileges on objets to user@host [identified by 'pass'] [with grant option]
```

Les clauses `identified by` et `with grant option` sont optionelles. La première permet de préciser un mot de passe pour l'utilisateur, la seconde l'autorise à donner ses privilèges à un autre utilisateur.

La liste *`privileges`* peut être remplacée par le mot « `all` » pour signifier tous les privilèges. Les privilèges sont séparés par des virgules et proviennent de cette liste :

```
select insert update delete create drop references index alter
```

à laquelle est ajoutée celle-ci pour les privilèges au niveau global (`on *.*`) :

```
reload shutdown process file
```

La liste *`objets`* peut prendre l'une de ces valeurs :

- « `*.*` » donne des privilèges au niveau global (table `user`) ;
- « *`database`*`.*` » affecte toutes les tables d'une base de données (table `db`) ;
- « *`database`*`.`*`table`* » affecte une table particulière (table `tables_priv`).

Droits d'accès, `grant`

Par exemple, un script de création d'une base de données commence souvent ainsi :

```
create database pizzas;
grant select, insert, update, delete on pizzas.*
  to pizzeria@localhost identified by 'italia';
use pizzas;
create table...
```

Ce script est lancé avec l'utilisateur d'administration du serveur MySQL :

```
mysql -u root -p mysql < cr_pizzas.sql
```

Ceci permet de créer un utilisateur spécifique avec des droits qui lui permettent uniquement d'utiliser le LMD afin de manipuler les données dans les tables que l'administrateur aura créées pour lui.

Droits d'accès, `revoke, set password`

La commande revoke permet de révoquer les droits d'un utilisateur.

Sa syntaxe est :

```
revoke privileges on objets from user@host
```

La liste des privilèges et des objets sont utilisées de la même manière qu'avec la commande `grant`.

La commande `set password` permet de modifier le mot de passe d'un utilisateur.

Sa syntaxe est :

```
set password for user@host = 'password'
```

Elle est souvent utilisée avec la fonction `password` qui permet de crypter un mot de passe en clair vers le format utilisé par MySQL.
Par exemple :

```
set password for pizzeria@localhost = password('PastaFantastica');
```

Droits d'accès, `show grants`

Pour en terminer avec la gestion des droits d'accès, l'une des options de la commande `show` permet d'obtenir une liste synthétique, mais exhaustive, des privilèges qui ont été octroyés à un utilisateur.

Sa syntaxe est :

```
show grants for user@host
```

Par exemple :

```
mysql> show grants for pizzeria@localhost;
+-----------------------------------------------------------------------------------------+
| Grants for pizzeria@localhost                                                           |
+-----------------------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'pizzeria'@'localhost' IDENTIFIED BY PASSWORD '72de524a554dc726' |
| GRANT SELECT, INSERT, UPDATE, DELETE ON pizzas.* TO 'pizzeria'@'localhost'              |
+-----------------------------------------------------------------------------------------+
2 rows in set (0.00 sec)
```

Fonctions

MySQL met à disposition un ensemble de fonctions.

Ces fonctions peuvent être utilisées dans les clauses `select`, `where`, `update` du LMD. Elle sont traditionnellement rangées selon ces catégories : numériques, chaînes, dates, générales et agrégeantes.

Consulter la référence de MySQL pour obtenir la description précise de ces fonctions (par exemple ici : `http://dev.nexen.net/docs/mysql/annotee/functions.php`).

Seules quelques fonctions sont présentées sous forme d'exemples dans les pages qui suivent.

Fonctions

```
select user();
-> pizzeria@localhost

select substring_index( user(), '@', 1 );
-> pizzeria

select version();
-> 3.23.49-log

select sum(prix_total), round( sum(prix_total), 2 ) from commandes;
-> 167.59999847412  167.60

select concat( 'Pizza ', nom ) from pizzas;
-> Pizza Reine
   Pizza Roi
   Pizza Végétarienne
```

Fonctions

```
update commandes set date_livraison = now() where id = 11;

select dayofweek(date_commande) as jour, count(*) as commandes
  from commandes group by dayofweek(date_commande) order by 2 desc;
-> jour  commandes
      7          4
      6          2
      1          2
      2          1
      3          1
```

(1=dimanche, 2=lundi, ..., 7=samedi)

```
select nom, ifnull(note, '-pas de commentaire-') as note from clients;
-> nom                 note
   David Solomon       -pas de commentaire-
   Linda Tchaïkowsky   -pas de commentaire-
   Arthur Bab's        Trés bon client
   Alex Feinberg       Mauvais payeur
```

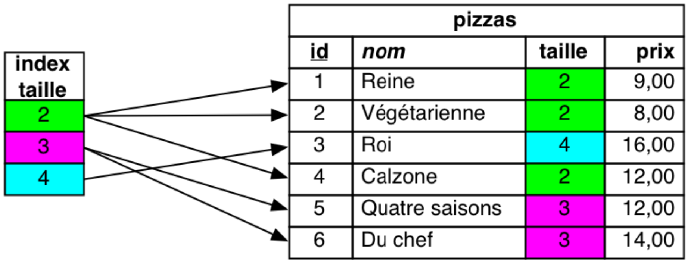
Index

Lorsque serveur MySQL évalue une clause `where`, il doit parcourir l'ensemble des valeurs des colonnes concernées. Cette recherche est coûteuse en temps.

Afin d'améliorer la vitesse d'exécution des requêtes sur des tables importantes, on utilise des objets nommés index. Un index est une relation simple entre l'ensemble des valeurs définies sur une colonne et les lignes de la table qui contiennent ces valeurs.

L'index est mis-à-jour en temps réel lorsque les lignes de la table sont modifiées par des requêtes `insert`, `update` ou `delete`. Par conséquent, plus il y a d'index sur une table, moins l'enregistrement des modifications est rapide. Mais le gain en performance sur la clause `where` est tel que les index restent très efficaces dans presque toutes les situations.

Il est possible de créer un index sur une ou plusieurs colonnes ou de limiter son contenu à des valeurs uniques (en fait, les clés primaires et uniques sont implémentées sous forme d'index).



pizzas

id	*nom*	taille	prix
1	Reine	2	9,00
2	Végétarienne	2	8,00
3	Roi	4	16,00
4	Calzone	2	12,00
5	Quatre saisons	3	12,00
6	Du chef	3	14,00

Index

Les index sont spécifiés lors de la création d'une table ou en utilisant le verbe `alter table`. Par ailleurs, des index implicites sont créés lorsque des clés primaires ou uniques sont utilisées.

Par exemple :

```
create table livres
( id             integer unsigned not null primary key
, code_isbn      varchar(20)      not null unique
, titre          varchar(200)     not null
, auteur_id      integer unsigned not null
, date_parution  date             not null
, resume         text
, index (titre)
, index (auteur_id)
, index (date_parution) );
```

Cette requête de création de table va générer cinq index en tout.
Utiliser beaucoup d'index est cohérent dans ce cas car le nombre de modification qui auront lieu sur cette table sera probablement bien inférieur au nombre de requêtes.

Index

La commandes `show index` permet d'obtenir la liste de tous les index d'une table.

Par exemple :

```
mysql> show index from livres;
+--------+------------+--------------+--------------+--------------+-----------+-------------+----------+--------+---------+
| Table  | Non_unique | Key_name     | Seq_in_index | Column_name  | Collation | Cardinality | Sub_part | Packed | Comment |
+--------+------------+--------------+--------------+--------------+-----------+-------------+----------+--------+---------+
| livres |          0 | PRIMARY      |            1 | id           | A         |           0 |     NULL | NULL   |         |
| livres |          0 | code_isbn    |            1 | code_isbn    | A         |           0 |     NULL | NULL   |         |
| livres |          1 | titre        |            1 | titre        | A         |        NULL |     NULL | NULL   |         |
| livres |          1 | auteur_id    |            1 | auteur_id    | A         |        NULL |     NULL | NULL   |         |
| livres |          1 | date_parution |            1 | date_parution | A         |        NULL |     NULL | NULL   |         |
+--------+------------+--------------+--------------+--------------+-----------+-------------+----------+--------+---------+
5 rows in set (0.00 sec)
```

Il est possible de nommer les index et les clés uniques lors de leur création :

```
create table livres
.../...
, unique code_isbn_uk (code_isdn)
, index titre_idx (titre)
.../...
```

Index

Le verbe `alter table` permet de supprimer ou de créer des index sur une table a posteriori.

```
alter table table add index [nom_index] (colonnes)
alter table table add unique [nom_index] (colonnes)
alter table table drop index nom_index
alter table table drop primary key
```

Il existe également ces formes équivalentes :

```
create [unique] index [nom_index] on table (colonnes)
drop index nom_index on table
```

Colonnes `auto_increment`

Très souvent les clés primaires des tables sont constituées d'une seule colonne dont le type est un entier. Il est alors très intéressant d'utiliser l'option `auto_increment` pour cette colonne lors de la création de la table.

Lors de l'insertion, si une colonne possède cette option et si la valeur qui y est insérée est NULL, MySQL affecte automatiquement la prochaine valeur à cette colonne.

Par exemple :

```
create table messages
( id     integer unsigned not null auto_increment primary key
, texte  text             not null );

insert into messages (texte) values ('Hello World!');
insert into messages values (NULL, 'Salut et merci pour le poisson');
select * from messages;
->  id   texte
     1   Hello World!
     2   Salut et merci pour le poisson
```

Colonnes `auto_increment`

Il ne peut y avoir qu'une seule colonne `auto_increment` dans une table et cette colonne doit faire partie de la clé primaire.

La fonction `last_insert_id` est utile pour connaître la valeur qui a été données à cette colonne lors de l'insertion.

Par exemple :

```
insert into messages (texte) values ('Et pourtant elle tourne');
select last_insert_id();
+------------------+
| last_insert_id() |
+------------------+
|                3 |
+------------------+
1 row in set (0.00 sec)
```

Index fulltext

Il est possible de créer des index « fulltext » sur les colonnes de type `char`, `varchar` ou `text` (et de ses dérivés : `tinytext`, `mediumtext` et `longtext`).

Ces index permettent de réaliser des recherches très rapides de mots sur des textes longs. L'algorithme utilise une technique dite de « scoring ». Par exemple (tiré directement de la page `http://dev.nexen.net/docs/mysql/annotee/fulltext-search.php`) :

```
create table articles
( id       integer unsigned not null auto_increment primary key
, titre    varchar(200)     not null
, contenu  text             not null
, fulltext (titre, contenu) );
```

Les deux index suivants ont été créés :

```
mysql> show index from articles;
+----------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+----------+
| Table    | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Comment  |
+----------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+----------+
| articles |          0 | PRIMARY  |            1 | id          | A         |           0 |     NULL | NULL   |          |
| articles |          1 | titre    |            1 | titre       | A         |        NULL |     NULL | NULL   | FULLTEXT |
| articles |          1 | titre    |            2 | contenu     | A         |        NULL |        1 | NULL   | FULLTEXT |
+----------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+----------+
3 rows in set (0.00 sec)
```

Index fulltext

```
mysql> select * from articles;
+----+------------------------------+----------------------------------------+
| id | titre                        | contenu                                |
+----+------------------------------+----------------------------------------+
|  0 | MySQL Tutorial               | DBMS stands for DataBase ...           |
|  1 | How To Use MySQL Efficiently | After you went through a ...           |
|  2 | Optimising MySQL             | In this tutorial we will show ...      |
|  3 | 1001 MySQL Tricks            | 1. Never run mysqld as root. 2. ...    |
|  4 | MySQL vs. YourSQL            | In the following database comparison ... |
|  5 | MySQL Security               | When configured properly, MySQL ...    |
+----+------------------------------+----------------------------------------+
6 rows in set (0.00 sec)

mysql> select *, match (titre, contenu) against ('database') as score
    -> from articles where match (titre, contenu) against ('database');
+----+-------------------+----------------------------------------+------------------+
| id | titre             | contenu                                | score            |
+----+-------------------+----------------------------------------+------------------+
|  4 | MySQL vs. YourSQL | In the following database comparison ... | 0.66266459031789 |
|  0 | MySQL Tutorial    | DBMS stands for DataBase ...           | 0.65545834044456 |
+----+-------------------+----------------------------------------+------------------+
2 rows in set (0.00 sec)
```

Références

Livres

Aux éditions O'Reilly :

« Pratique de MySQL et PHP » (http://www.oreilly.fr/catalogue/2841772373.html)
« Managing & Using MySQL, 2nd edition » (http://www.oreilly.com/catalog/msql2)
« MySQL Pocket Reference » (http://www.oreilly.com/catalog/mysqlpr)
« MySQL Cookbook » (http://www.oreilly.com/catalog/mysqlckbk)
« MySQL Reference Manual » (http://www.oreilly.com/catalog/mysqlref)

Aux éditions OEM (Osman Eyrolles Multimédia) :

« MySQL » (http://www.editions-eyrolles.com/php.informatique/Ouvrages/9782746404267.php3)

Sites Web

Site de MySQL : http://www.mysql.com
Documentation française annotée : http://dev.nexen.net/docs/mysql