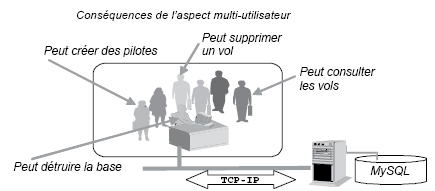
Contrôle des données

Comme dans tout système multi-utilisateur, l’usager d’un SGBD doit être identifié avant de pouvoir utiliser des ressources. Les accès aux informations et à la base de données doivent être contrôlés à des fins de sécurité et de cohérence. La figure suivante illustre un groupe d’utilisateurs dans lequel existe une classification entre ceux qui peuvent consulter, mettre à jour, supprimer des enregistrements, voire les tables.



Nous verrons dans cette section les aspects du langage SQL qui concernent le contrôle des données et des accès. Nous étudierons :

* la gestion des utilisateurs qui manipuleront des bases de données dans lequelles se trouvent des objets tels que des tables, index, séquences (pour l’instant implémentées par des colonnes AUTO\_INCREMENT), vues, procédures, etc. ;
* la gestion des privilèges qui permettent de donner des droits sur la base de données (privilèges système) et sur les données de la base (privilèges objet) ;
* la gestion des vues ;
* l’utilisation du dictionnaire des données (base de données information\_schema).

# Gestion des utilisateurs

Un utilisateur (*user*) est identifié par MySQL par son nom et celui de la machine à partir de laquelle il se connecte. Cela fait, il pourra accéder à différents objets (tables, vues, séquences, index, procédures, etc.) d’une ou de plusieurs bases sous réserve d’avoir reçu un certain nombre de privilèges.

## Classification

Les types d’utilisateurs, leurs fonctions et leur nombre peuvent varier d’une base à une autre.

Néanmoins, pour chaque base de données en activité, on peut classifier les utilisateurs de la manière suivante :

* Le DBA (*DataBaseAdministrator*). Il en existe au moins un. Une petite base peut n’avoir qu’un seul administrateur. Une base importante peut en regrouper plusieurs qui se partagent les tâches suivantes :

– installation et mises à jour de la base et des outils éventuels ;

– gestion de l’espace disque et des espaces pour les données ;

– gestion des utilisateurs et de leurs objets (s’ils ne les gèrent pas eux-mêmes) ;

– optimisation des performances ;

– sauvegardes, restaurations et archivages ;

– contact avec le support technique.

* L’administrateur réseau (qui peut être le DBA) se charge de la configuration des couches client pour les accès distants.
* Les développeurs qui conçoivent et mettent à jour la base. Ils peuvent aussi agir sur leurs objets (création et modification des tables, index, séquences, etc.). Ils transmettent au DBA leurs demandes spécifiques (stockage, optimisation, sécurité).
* Les administrateurs d’application qui gèrent les données manipulées par la ou les applications. Pour les petites et les moyennes bases, le DBA joue ce rôle.
* Les utilisateurs qui se connectent et interagissent avec la base à travers les applications ou à l’aide d’outils (interrogations pour la génération de rapports, ajouts, modifications ou suppressions d’enregistrements).

Tous seront des utilisateurs (au sens MySQL) avec des privilèges différents.

## Création d’un utilisateur (CREATE USER)

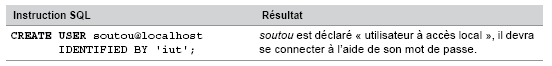
Pour pouvoir créer un utilisateur, vous devez posséder le privilège CREATE USER ou INSERT sur la base système mysql (car c’est la table mysql.user qui stockera l’existence de ce nouvel arrivant).

La syntaxe de création d’un utilisateur est la suivante :



IDENTIFIED BY *motdePasse* permet d’affecter un mot de passe (16 caractères maximum, sensibles à la casse) à un utilisateur (16 caractères maximum, sensibles aussi à la casse).

Le tableau suivant décrit la création d’un utilisateur (à exécuter en étant connecté en local en tant que root) :



Par défaut, les utilisateurs, une fois créés, n’ont aucun droit sur aucune base de données (à part en lecture écriture sur la base test et en lecture seule sur la base information\_schema ).

La section *Privilèges* étudie ces droits.

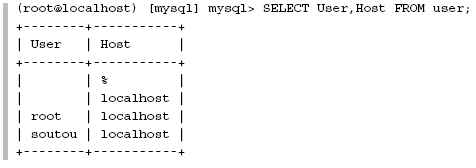
### Un utilisateur bien connu

Lors de l’installation, vous avez dû noter la présence de l’utilisateur root (mot de passe saisi à l’installation). Cet utilisateur est le DBA que MySQL vous offre. Il vous permettra d’effectuer vos tâches administratives en ligne de commande ou par une console graphique (créer des utilisateurs par exemple).

### Liste des utilisateurs

À propos de root, on le retrouve dans la table user de la base mysql (mysql.user).

L’extraction des colonnes User et Host restitue la liste des utilisateurs connus du serveur. Si root n’avait pas sélectionné la base mysql , la commande à exécuter aurait été « SELECT User,Host FROM mysql.user; ».



Vous devez posséder une table similaire. Il apparaît quatre accès potentiels. L’utilisateur vide ' ' correspond à une connexion anonyme. La machine désignée par « % » indique que la connexion est autorisée à partir de tout site (en supposant qu’un client MySQL est installé et qu’il est relié au serveur par TCP-IP). La machine désignée par « localhost» spécifie que la connexion est autorisée en local.

Ici, la table fait état que l’accès anonyme (restreint toutefois à la base test , voir la section *Table* mysql.db ) est permis en local et à partir de tout site, et que soutou comme root ne peuvent se connecter qu’en local.

### Modification d’un utilisateur

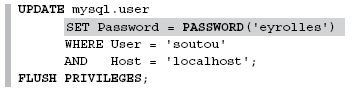
Le mot de passe d’un utilisateur peut être modifié sans parler de privilèges. Nous verrons plus tard qu’il est possible de restreindre le nombre de requêtes ( SELECT ), de modifications (UPDATE), de connexions par heure et de connexions simultanées à un serveur.

Puisqu’il n’existe pas de commande ALTER USER, pour changer un mot de passe, il faut donc modifier la table user par la seule commande SQL capable de le faire : UPDATE.

L’instruction suivante modifie le mot de passe de l’utilisateur soutou pour l’accès en local.

Notez l’utilisation de la fonction PASSWORD()qui code le mot de passe à affecter à la colonne Password de la table user. Il est plus prudent d’utiliser ensuite FLUSH PRIVILEGES qui recharge les tables système de manière à rendre la manipulation effective sur l’instant (un peu comme un

COMMIT sur des données).



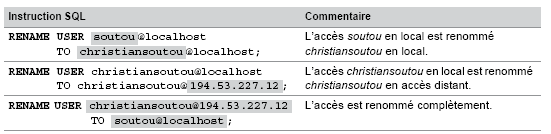
Une fois cette modification réalisée, si soutou tente une connexion avec son ancien mot de passe, il vient l’erreur classique : « ERROR 1045 (28000): Access denied for user 'soutou'@'localhost' (using password: xx) » (*xx* valant YES si soutou se connecte avec son ancien mot de passe, NO s’il n’en donne pas). Chaque utilisateur peut changer son propre mot de passe à l’aide de cette instruction s’il en a le privilège. Mais attention ! Le fait de lui donner ce droit (nous verrons plus loin comment le faire) implique également qu’il puisse aussi modifier les mots de passe de ses copains, ainsi que celui du root !

## Renommer un utilisateur (RENAME USER)

Pour pouvoir renommer un utilisateur, vous devez posséder le privilège CREATE USER (ou le privilège UPDATE sur la base de données mysql). La syntaxe SQL est la suivante :



Penser à spécifier l’accès complet à renommer (user@machine). Les privilèges et le mot de passe ne changent pas. Le tableau suivant décrit trois opérations de renommage d’utilisateurs (qui reviennent d’ailleurs à l’état initial).



## Suppression d’un utilisateur (DROP USER)

Pour pouvoir supprimer un utilisateur, vous devez posséder le privilège CREATE USER (ou le privilège DELETE sur la base de données mysql). La syntaxe SQL est la suivante :



Il faut spécifier l’accès à éliminer (user@machine). Tous les privilèges relatifs à cet accès sont détruits. Si l’utilisateur est connecté dans le même temps, sa suppression ne sera effective qu’à la fin de sa (dernière) session.

Aucune donnée d’aucune table que l’utilisateur aura mis à jour durant toutes ses connexions ne sera supprimée. Il n’y a pas de notion d’appartenance d’objets (tables, index, procédure, etc.) à un utilisateur. Tout ceci est relatif à la base de données (*database*). Pour supprimer le compte soutou en local, la commande à lancer est :



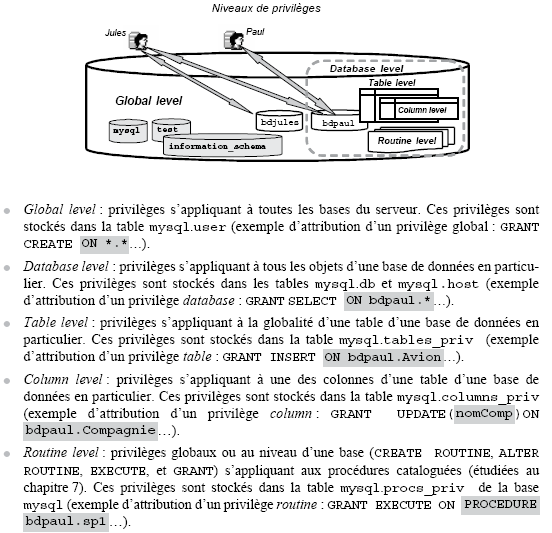
# Privilèges

Un privilège (sous-entendu *utilisateur*) est un droit d’exécuter une certaine instruction SQL (on parle de privilège *système*), ou un droit relatif aux données des tables situées dans différentes bases (on parle de privilège *objet*). La connexion, par exemple, sera considérée comme un privilège système bien que n’étant pas une commande SQL.

Les privilèges système diffèrent sensiblement d’un SGBD à un autre. Chez Oracle, il y en a plus d’une centaine, MySQL est plus modeste en n’en proposant qu’une vingtaine. En revanche, on retrouvera les mêmes privilèges objet (exemple : autorisation de modifier la colonne nomComp de la table Compagnie) qui sont attribués ou retirés par les instructions GRANT et REVOKE.

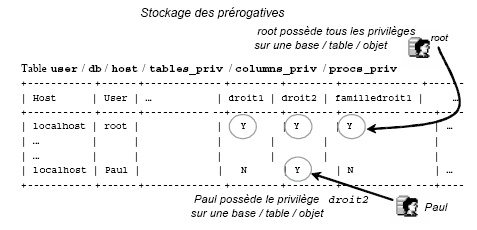
## Niveaux de privilèges

La figure suivante illustre les différents niveaux de privilèges que l’on peut rencontrer :



## Tables de la base mysql

Cinq tables de la base de données mysql suffisent à MySQL pour stocker les privilèges (système et objet) de tous les utilisateurs. La figure suivante illustre comment MySQL déduit toutes ces prérogatives toujours en fonction des accès (couple utilisateur, machine).



La colonne Db est en plus dans les tables host, tables\_priv et columns\_priv, car elle est nécessaire pour désigner la base de données sur laquelle portera le droit ou la famille de droits.

Supposons, pour nos exemples, que l’utilisateur Paul (accès en local) et la base de données bdpaul soient créés.

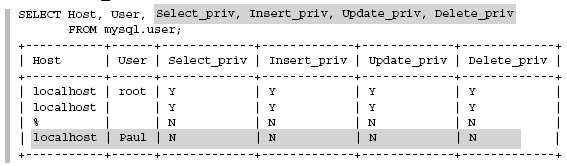


## Table mysql.user

Cette table est composée de 37 colonnes qui décrivent les privilèges au niveau global du serveur. Nous détaillons ici la signification des principales.

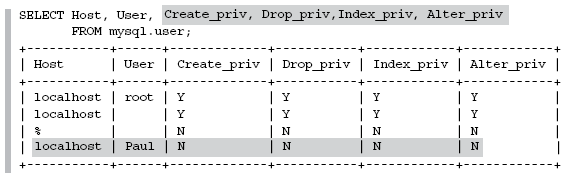
### Privilèges objet (LMD) sur toutes les bases de données

La requête suivante extrait les prérogatives de Paul (et des autres). Pour l’instant, le caractère 'N' étant dans toutes les colonnes, il ne peut ni interroger une table (Select\_priv), ni insérer dans une table (Insert\_priv), ni en modifier (Update\_priv), ni en supprimer (Delete\_priv), et ce quelle que soit la base de données (excepté les bases système test et information\_schema) sur laquelle il voudra se connecter.



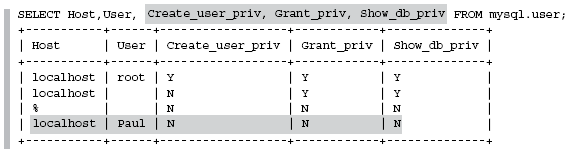
### Privilèges objet (LDD) sur toutes les bases de données

La requête suivante extrait les prérogatives à propos des instructions LDD. Pour l’instant, le caractère 'N' étant dans toutes les colonnes, Paul ne peut ni créer une table ou une base (Create\_priv), ni en supprimer (Drop\_priv), ni créer ou supprimer un index (Index\_priv), ni modifier la structure d’une table, la renommer ou modifier une base (Alter\_priv), et ce quelle que soit la base de données (excepté les bases système test et information\_schema).



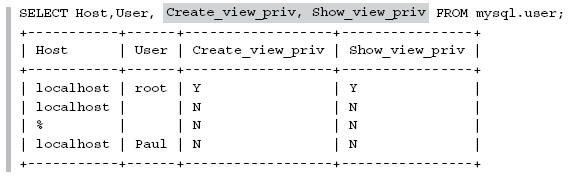
### Privilèges système (LCD) sur toutes les bases de données

La requête suivante extrait les prérogatives à propos des instructions LCD. Pour l’instant, le caractère 'N' étant dans toutes les colonnes, Paul ne peut ni créer un utilisateur (Create\_user\_priv), ni transmettre des droits qu’il aura lui-même reçus (Grant\_priv), ni lister les bases de données existantes (Show\_db\_priv), et ce quelle que soit la base de données.



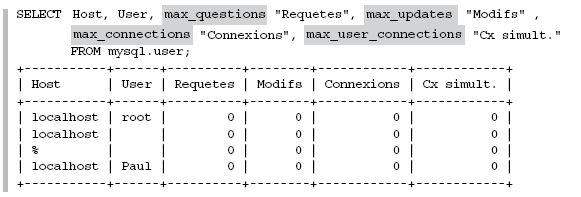
### Privilèges à propos des vues sur toutes les bases de données

La requête suivante extrait les prérogatives à propos des instructions relatives aux vues (*views* détaillées dans la section suivante). Pour l’instant, le caractère 'N' étant dans toutes les colonnes, Paul ne peut ni créer une vue (Create\_view\_priv), ni lister les vues existantes (Show\_view\_priv), et ce quelle que soit la base de données.



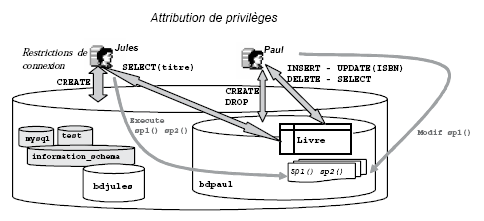
### Privilèges à propos des restrictions d’utilisateur

La requête suivante extrait les prérogatives à propos des restrictions qu’on peut définir par accès. Pour l’instant, le chiffre étant à 0 dans toutes les colonnes, aucun accès (utilisateur) n’est limité concernant le nombre de requêtes (max\_questions), de modifications (max\_updates), de connexions par heure (max\_connections) et de connexions simultanées (max\_user\_connections) à un serveur.



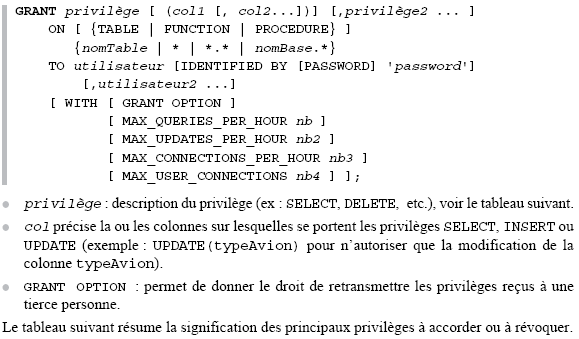
## Attribution de privilèges (GRANT)

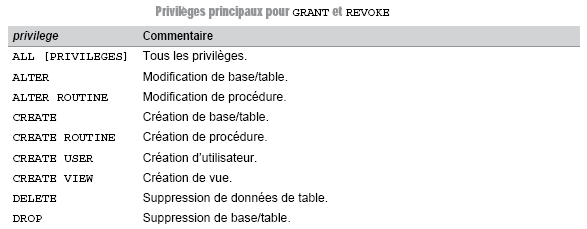
La figure suivante illustre le contexte qui va servir d’exemple à l’attribution de prérogatives.

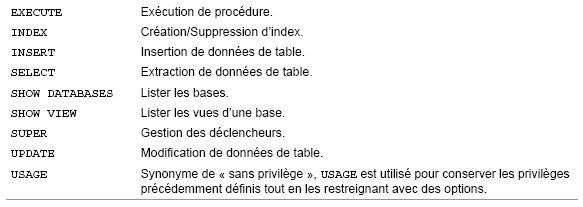


### Syntaxe

L’instruction GRANT permet d’attribuer un (ou plusieurs) privilège(s) à propos d’un objet à un (ou plusieurs) bénéficiaire(s). L’utilisateur qui exécute cette commande doit avoir reçu lui-même le droit de transmettre ces privilèges (reçu avec la directive GRANT OPTION). Dans le cas de root, aucun problème, car il a implicitement tous les droits.

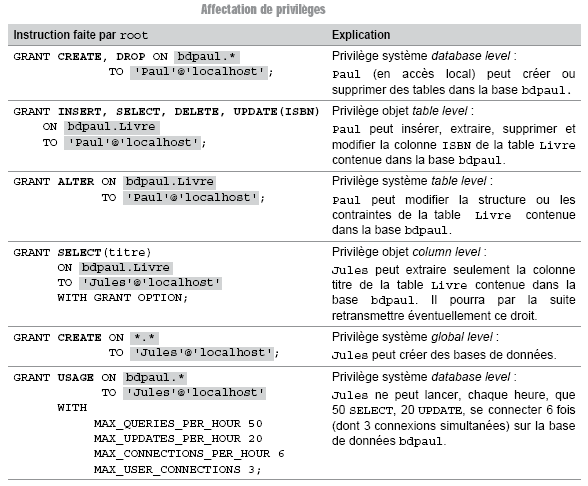






### Exemples

Le tableau suivant décrit l’affectation de quelques privilèges en donnant les explications associées.



Tout ce que vous avez le droit de faire doit être explicitement autorisé par la commande GRANT.

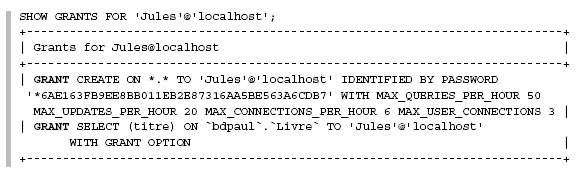
Ce qui n’est pas dit par GRANT n’est pas permis. Par exemple, Jules peut créer des bases, mais pas en détuire, Paul peut modifier le numéro ISBN d’un livre mais pas son titre, etc.

### Voir les privilèges

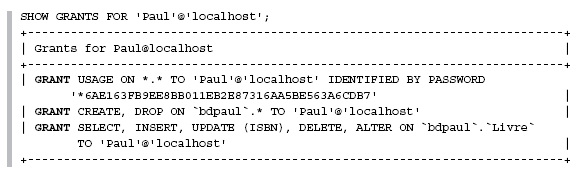
La commande SHOW GRANTS FOR liste les différentes instructions GRANT équivalentes à toutes les prérogatives d’un utilisateur donné. C’est bien utile quand vous avez attribué un certain nombre de privilèges à un utilisateur sans avoir pensé à les consigner dans un fichier de commande.



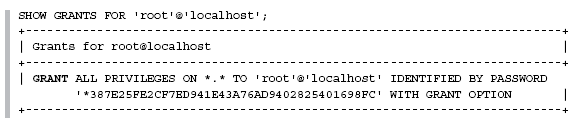
Utilisons cette commande pour extraire les profils de Jules, de Paul et de l’administrateur en chef (accès en local). J’avoue avoir un peu retravaillé l’état de sortie (sans en modifier une ligne quand même).



On remarque que MySQL a regroupé deux privilèges en une instruction GRANT (CREATE et les restrictions de connexions). Par là même, on se rend compte que les prérogatives de connexion sont au niveau *global*, bien qu’on les ait spécifiées au niveau *database*.

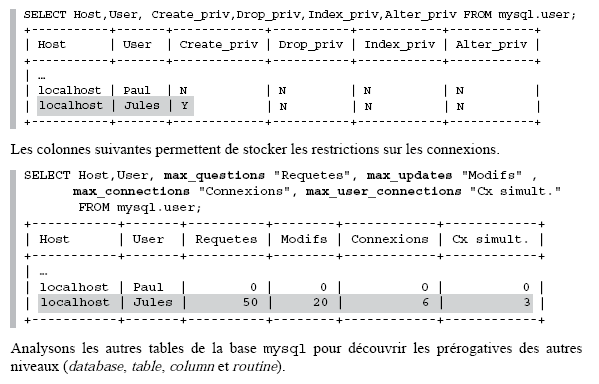


On remarque que MySQL a regroupé tous les privilèges sur la table Livre en une instruction GRANT. La première exprime le fait que Paul peut se connecter à toutes les bases (par USE nomBase), mais qu’il ne pourra travailler en réalité que dans bdpaul.



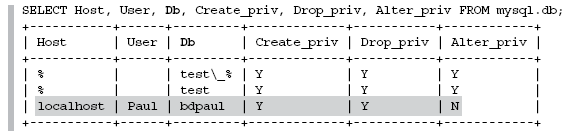
On remarque que MySQL n’attribue qu’un seul droit, mais le plus fort ! Tous les droits (ALL PRIVILEGES) sur toutes les bases (\*.\*), avec en prime la clause GRANT OPTION qui permet de retransmettre n’importe quoi à n’importe qui, ou de tout révoquer.

Interrogeons à nouveau la table user de la base mysql stockant les prérogatives au niveau global du moment. Le droit de création en local de Jules apparaît sur toutes les bases.



## Table mysql.db

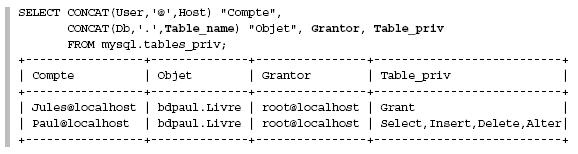
La table mysql.db décrit les prérogatives au niveau *database*. Ainsi la colonne Db indique la base de données.



Notez la possibilité de Paul, avec l’accès local, de créér et de supprimer des tables dans la base bdpaul. Notez également la possibilité de créer, de supprimer, de modifier des tables par un accès distant anonyme sur la base test.

## Table mysql.tables\_priv

La table mysql.tables\_priv décrit les prérogatives objet au niveau *table*. Ainsi la colonne Table\_name indique la table concernée, la colonne Grantor précise l’utilisateur ayant donné le droit. La colonne Table\_priv est un SET contenant la liste des droits de l’utilisateur sur la table.

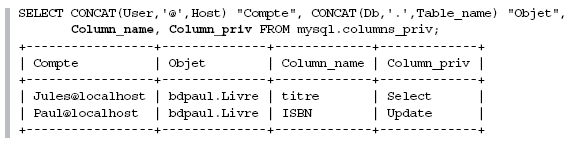


On retrouve les quatre privilèges de Paul et celui de Jules (GRANT OPTION de SELECT sur la table).

Cette table possède aussi une colonne de nom Timestamp stockant l’instant au cours duquel s’est déroulée l’attribution (ou la révocation).

## Table mysql.columns\_priv

La table mysql.columns\_priv décrit les prérogatives objet au niveau *column*. Ainsi la colonne Table\_name indique la table concernée, la colonne Column\_name précise la colonne concernée par le droit. La colonne Column\_priv est un SET contenant la liste des droits de l’utilisateur sur la colonne de la table.



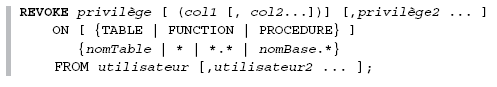
On retrouve le privilège de Paul et celui de Jules (portant ici sur la même table).

## Révocation de privilèges (REVOKE)

La révocation d’un ou de plusieurs privilèges est réalisée par l’instruction REVOKE. Pour pouvoir révoquer un privilège, vous devez détenir (avoir reçu) au préalable ce même privilège avec l’option WITH GRANT OPTION.

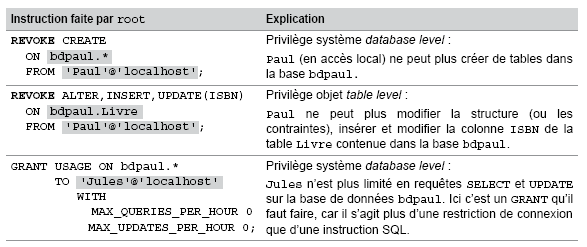
### Syntaxe

Dans la syntaxe suivante, les options sont les mêmes que pour la commande GRANT.



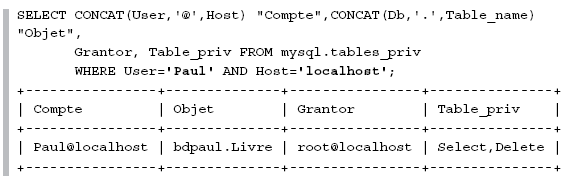
### Exemples

Le tableau suivant décrit la révocation de certains privilèges acquis des utilisateurs Paul et Jules.

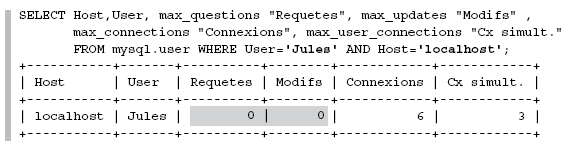


### Vérifications

Une fois ces actualisations réalisées, les cinq tables de la base mysql contiennent un peu plus le caractère 'N' qu’auparavant. Les colonnes SET des tables mysql.tables\_priv, mysql.columns\_priv et mysql.procs\_priv sont également mises à jour. Ainsi, l’extraction du profil actuel de Paul au niveau *table* fait apparaître les deux seuls droits qu’il lui reste.



L’extraction du profil actuel de Jules au niveau *database* fait apparaître que les deux limitations de connexion sur les SELECT et UPDATE ont disparu.



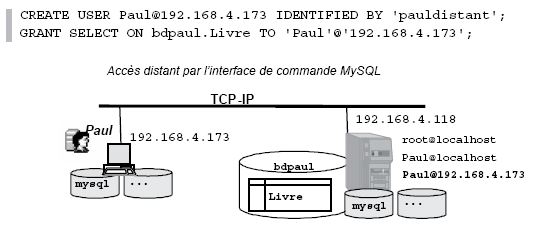
### Tout en une fois !

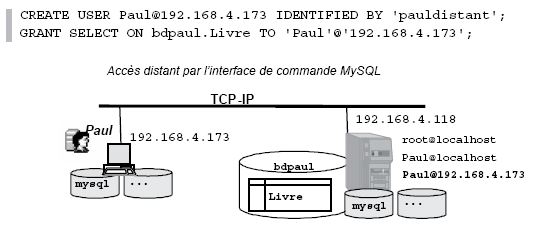
Il existe une instruction qui révoque tous les droits en une fois.



# Accès distants

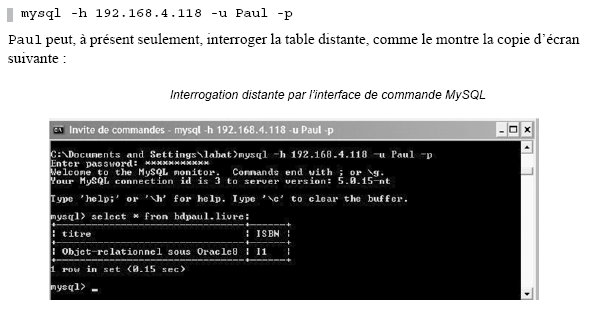
La figure suivante illustre la configuration du test. Un client est en 192.168.4.173 sur lequel sont installées les couches MySQL (*Complete Package* ou *Essentials Package*). Un serveur est en 192.168.4.118 équipé de MySQL *Complete Package*. Sur le serveur, root crée un accès à Paul, en précisant l’adresse de la machine client, et lui attribue un droit d’extraction de la table Livre sur la base bdPaul.





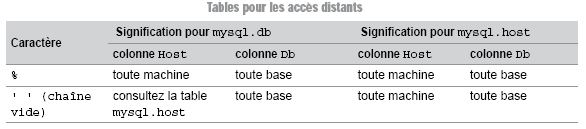
## Connexion par l’interface de commande

Sur le client, Paul se connecte au serveur dans une fenêtre de commande, en précisant l’adresse de la machine serveur, puis donne son mot de passe distant. Pensez à enlever les pare-feu Windows sur le client et le serveur (bloquant le port 3306).



## Table mysql.host

La table mysql.host est utilisée conjointement avec mysql.db et concerne les accès distants (plusieurs machines). Cette table n’est employée que pour les prérogatives au niveau *database*, indépendamment des utilisateurs. La structure est la même que celle de mysql.db, à l’exception de la colonne User qui n’est pas présente. Le couple de colonnes (Host, Db) est unique.



MySQL lit et trie les tables db (sur les colonnes Host, Db et User) et host (sur les colonnes

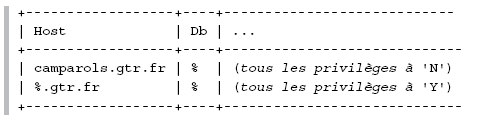
Host et Db) en même temps qu’il parcourt la table user. Pour les opérations relatives aux bases (INSERT, UPDATE, etc.), MySQL interroge la table user. Si l’accès n’y est pas décrit, la recherche se poursuit dans les tables db et host. Si la colonne Host de la table db est renseignée en fonction de l’accès, l’utilisateur reçoit ses privilèges.

Si la colonne Host de la table db n’est pas renseignée (' '), cela signifie que la table host énumère les machines qui sont autorisées à accéder à une base de données en particulier. Si la machine ne correspond pas, l’accès n’est pas permis. Dans le cas contraire, les privilèges sont valués à 'Y' à partir d’une intersection (et pas d’une union) entre les tables db et host sur le couple (Host, Db).

La table mysql.host n’est mise à jour ni par GRANT, ni par REVOKE. Il faudra directement insérer (par INSERT), modifier (par UPDATE) ou supprimer (par DELETE) les lignes de cette table.

Elle n’est pas utilisée par la plupart des serveurs MySQL, car elle est dédiée à des usages très spécifiques (pour gérer un ensemble de machines à accès sécurisé, par exemple). Elle peut aussi être utilisée pour définir un ensemble de machines à accès non sécurisé.

En supposant que vous déclariez une machine à accès non sécurisé : camparols.gtr.fr. Il est possible d’autoriser l’accès sécurisé à toutes les autres machines du réseau local. Ceci en ajoutant des enregistrements par INSERT dans la table mysql.host comme suit :



Vous déclareriez l’inverse des conditions initiales en remplaçant les 'N' par des 'Y', et réciproquement.

Dans tous les cas, il sera nécessaire de mettre à jour les autres tables pour affiner les privilèges.

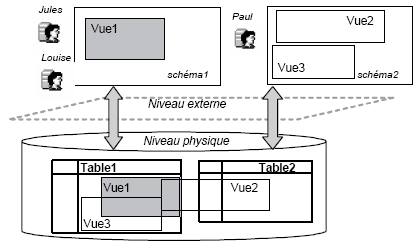
# Vues

Outre le contrôle de l’accès aux données (privilèges), la confidentialité des informations est un aspect important qu’un SGBD relationnel doit prendre en compte. Depuis la version 5 de MySQL, la confidentialité est renforcée par l’utilisation de vues *(views)* qui agissent comme des fenêtres sur la base de données. Cette section décrit les différents types de vues qu’on peut rencontrer.

Les vues correspondent à ce qu’on appelle « le niveau externe » qui reflète la partie visible de la base de données pour chaque utilisateur.

Seules les tables contiennent des données et, pourtant, pour l’utilisateur, une vue apparaît comme une table. En théorie, les utilisateurs ne devraient accéder aux informations qu’en questionnant des vues. Ces dernières masquant la structure des tables interrogées. En pratique, la plupart des applications se passent de ce concept en manipulant directement les tables.

La figure suivante illustre ce qui a été dit en présentant trois utilisateurs. Ils travaillent chacun sur une base de données contenant des vues formées à partir de différentes tables.



Une vue est considérée comme une table virtuelle car elle n’a pas d’existence propre. Seule sa structure est stockée dans le dictionnaire. Ses données seront extraites de la mémoire à partir des tables source, à la demande.

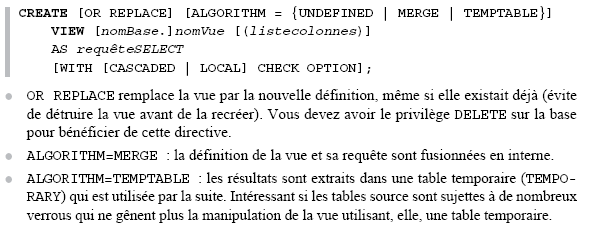
Une vue est créée à l’aide d’une instruction SELECT appelée « requête de définition ». Cette requête interroge une (ou plusieurs) table(s) ou vue(s). Une vue se recharge chaque fois qu’elle est interrogée.

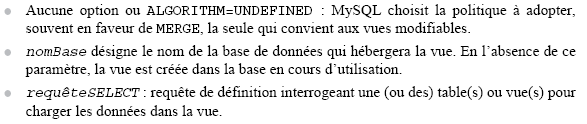
Outre le fait d’assurer la confidentialité des informations, une vue est capable de réaliser des contrôles de contraintes d’intégrité et de simplifier la formulation de requêtes complexes.

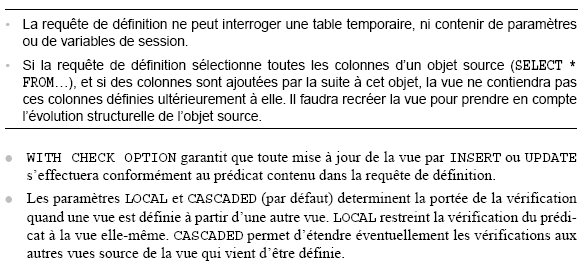
Dans certains cas, la définition d’une vue temporaire est nécessaire pour écrire une requête qu’il ne serait pas possible de construire à partir des tables seules. Attribuées comme des privilèges (GRANT), les vues améliorent la sécurité des informations stockées.

## Création d’une vue (CREATE VIEW)

Pour pouvoir créer une vue dans une base, vous devez posséder le privilège CREATE VIEW et les privilèges en SELECT des tables présentes dans la requête de définition de la vue. La syntaxe SQL de création d’une vue est la suivante :

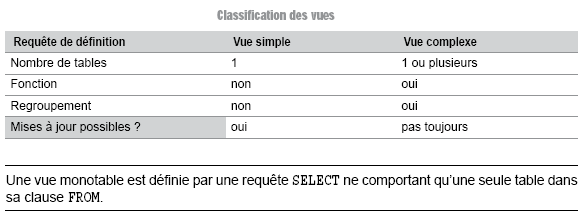






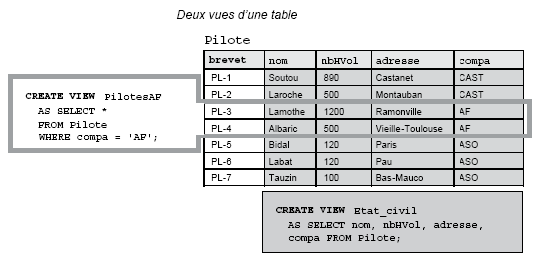
## Classification

On distingue les vues simples des vues complexes en fonction de la nature de la requête de définition. Le tableau suivant résume ce que nous allons détailler au cours de cette section :

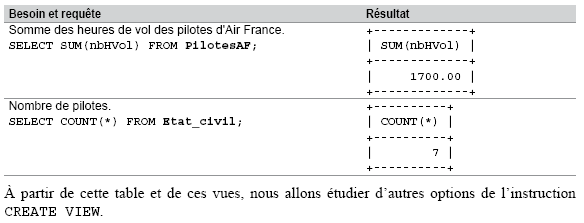


## Vues monotables

Les mécanismes présentés ci-après s’appliquent aussi, pour la plupart, aux vues multitables (étudiées plus loin). Considérons les deux vues illustrées par la figure suivante et dérivées de la table Pilote. La vue PilotesAF décrit les pilotes d’Air France à l’aide d’une restriction (éléments du WHERE). La vue Etat\_civil est constituée par une projection de certaines colonnes (éléments du SELECT).



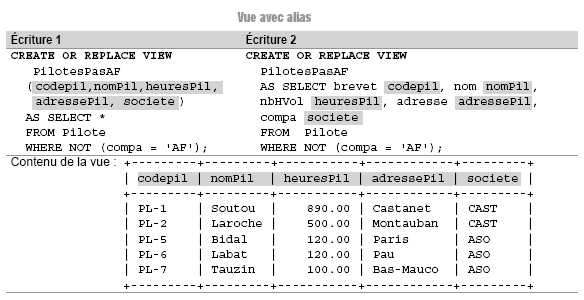
Une fois créée, une vue s’interroge comme une table par tout utilisateur, sous réserve qu’il ait obtenu le privilège en lecture directement (GRANT SELECT ON *nomVue* TO…). Le tableau suivant présente une interrogation des deux vues :

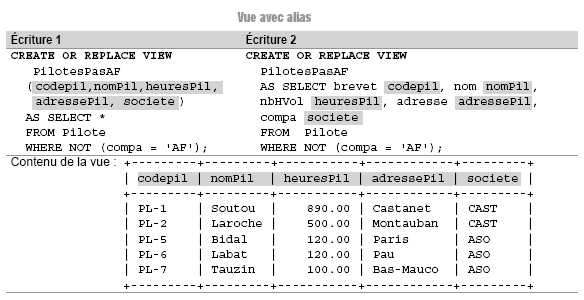


### Alias

Les alias, s’ils sont utilisés, désignent le nom de chaque colonne de la vue. Ce mécanisme permet de mieux contrôler les noms de colonnes. Quand un alias n’est pas présent, la colonne prend le nom de l’expression renvoyée par la requête de définition. Ce mécanisme sert à masquer les noms des colonnes de l’objet source.

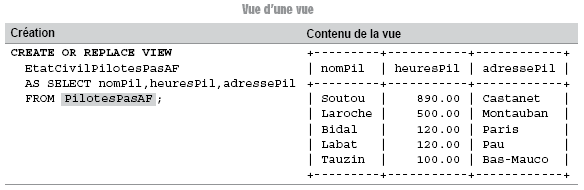
Les vues suivantes sont créées avec des alias qui masquent le nom des colonnes de la table source. Les deux écritures sont équivalentes.





### Vue d’une vue

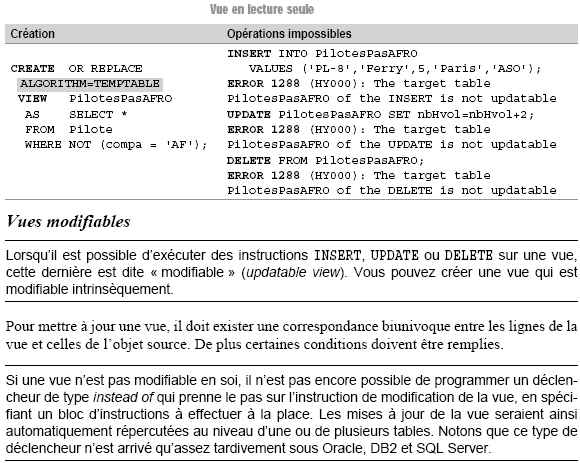
L’objet source d’une vue est en général une table, mais peut aussi être une vue. La vue suivante est définie à partir de la vue PilotesPasAF précédemment créée. Notez qu’il aurait été possible d’utiliser des alias pour renommer les colonnes de la nouvelle vue.

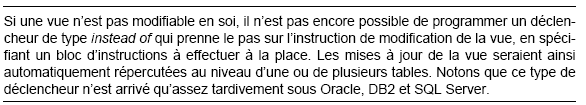


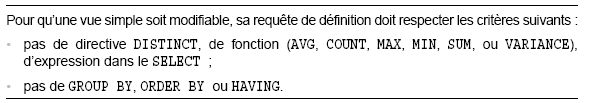
### Vues en lecture seule

L’option ALGORITHM=TEMPTABLE déclare la vue non modifiable par INSERT, UPDATE, ou DELETE.

Redéfinissons une vue des pilotes n’étant pas d’Air France à l’aide de cette option. Les messages d’erreur induits par la clause de lecture seule, générés par MySQL, sont très parlants.



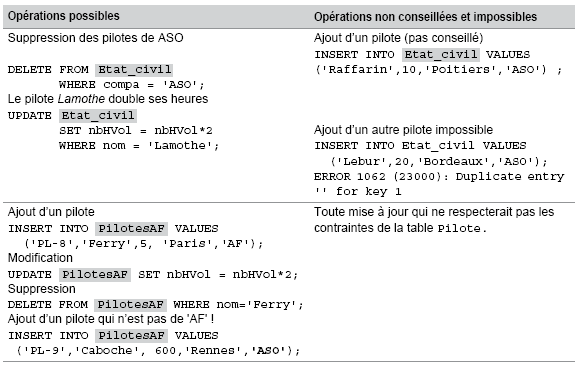




Dans notre exemple, nous constatons qu’il sera quand même possible d’ajouter un pilote à la vue Etat\_civil, bien que la clé primaire de la table source ne soit pas renseignée. MySQL insère à la place, en l’absence de valeur par défaut de la clé primaire, la chaîne vide (' ') ; si la clé avait été une séquence, les insertions se feraient normalement. Cette opération ne pourra donc se faire qu’une seule fois, après, cela sera contradictoire avec la condition de correspondance biunivoque.

En revanche, il sera possible de modifier les colonnes de cette vue. On pourra aussi ajouter, modifier (sous réserve de respecter les éventuelles contraintes issues des colonnes de la table source), ou supprimer des pilotes en passant par la vue PilotesAF.

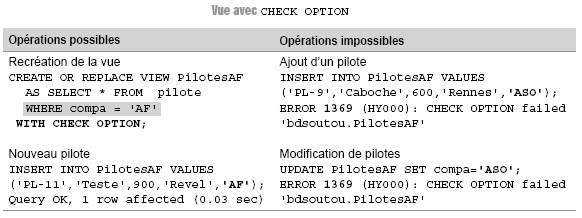
La dernière instruction est paradoxale, car elle permet d’ajouter un pilote de la compagnie 'ASO' en passant par la vue des pilotes de la compagnie 'AF'. La directive WITH CHECK OPTION permet d’éviter ces effets de bord indésirables pour l’intégrité de la base.



### Directive CHECK OPTION

La directive WITH CHECK OPTION empêche un ajout ou une modification non conformes à la définition de la vue.

Interdisons l’ajout (ou la modification de la colonne compa) d’un pilote au travers de la vue PilotesAF, si le pilote n’appartient pas à la compagnie de code 'AF'. Il est nécessaire de redéfinir la vue PilotesAF. Le script suivant décrit la redéfinition de la vue, l’ajout d’un pilote et les tentatives d’ajout et de modification ne respectant pas les caractéristiques de la vue. Les messages sont très clairs.



### Vues complexes

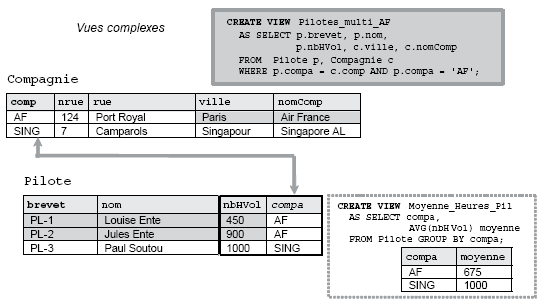
Une vue complexe est caractérisée par le fait qu’elle contient, dans sa définition, plusieurs tables (jointures) et une fonction appliquée à des regroupements ou à des expressions. La mise à jour de telles vues n’est pas toujours possible.

Pour pouvoir modifier une vue complexe, les restrictions sont les suivantes :

• La requête de définition ne doit pas contenir de sous-interrogation (jointure procédurale).

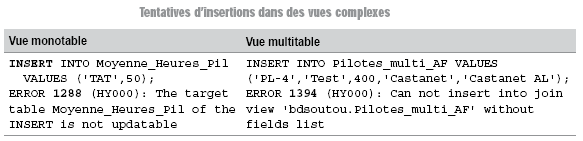
• Il n’est pas possible d’utiliser d’opérateur ensembliste (sauf UNION [ALL]).

La figure suivante présente deux vues complexes qui ne sont pas modifiables. La vue multitable Pilotes\_multi\_AF est créée à partir d’une jointure entre les tables Compagnie et Pilote. La vue Moyenne\_Heures\_Pil est créée à partir d’un regroupement de la table Pilote.



### Mises à jour

Il semblerait qu’on ne puisse insérer aucun enregistrement dans ces vues du fait de la cohérence qu’il faudrait établir du sens *vue vers tables*. Les messages d’erreur générés par MySQL sont différents suivant la nature de la vue (monotable ou multitable). Nous verrons comment résoudre l’erreur du deuxième cas.



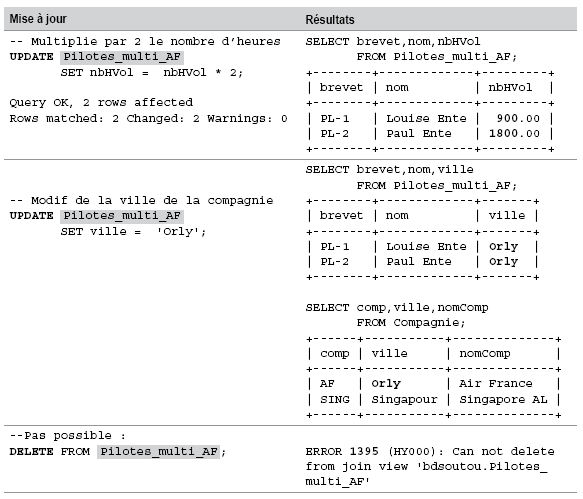
On pourrait croire qu’il en est de même pour les modifications. Ce n’est pas le cas. Alors que la vue monotable Moyenne\_Heures\_Pil n’est pas modifiable, ni par UPDATE ni par DELETE (message d’erreur 1288), la vue multitable Pilotes\_multi\_AF est transformable dans une certaine mesure, car la table Pilote (qui entre dans sa composition) est dite « protégée par clé » (*key preserved*). Nous verrons dans le prochain paragraphe la signification de cette notion.

Les colonnes de la vue correspondant à la table protégée par clé ne sont pas les seules à pouvoir être modifiées. Ici, nbHVol peut être mise à jour car elle appartient à la table protégée ; ville qui n’appartient pas à une table protégée peut aussi être modifée !

Les suppressions ne se répercutent pas sur les enregistrements de la table protégée par clé (Pilote).

Modifions et tentons de supprimer des enregistrements à travers la vue multitable Pilotes\_

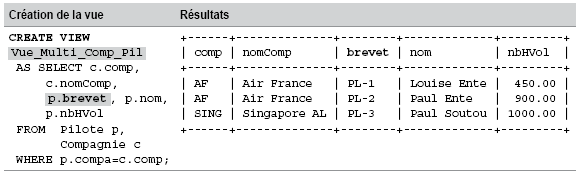
multi\_AF.



### Tables protégées (key preserved tables)

Une table est dite protégée par sa clé (*key preserved*) si sa clé primaire est préservée dans la clause de jointure et se retrouve en tant que colonne de la vue multitable (elle peut jouer le rôle de clé primaire de la vue).

En considérant les données initiales pour la vue multitable Vue\_Multi\_Comp\_Pil, la table préservée est la table Pilote, car la colonne brevet identifie chaque enregistrement extrait de la vue, alors que la colonne comp ne le fait pas.



Cela ne veut pas dire pour autant que cette vue est modifiable. Étudions à présent les conditions qui régissent ces limitations.

### Critères

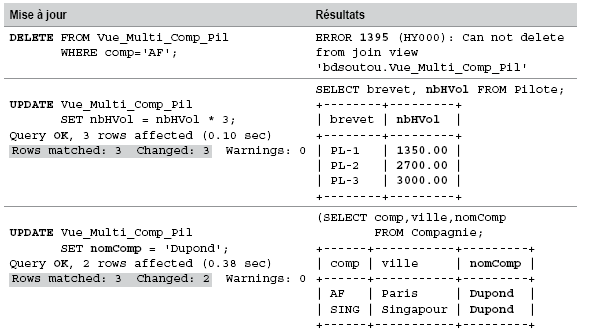
Une vue multitable modifiable (*updatable join view* ou *modifiable join view*) est une vue qui n’est pas définie avec l’option ALGORITHM=TEMPTABLE et qui est telle que la requête de définition contient plusieurs tables dans la clause FROM.

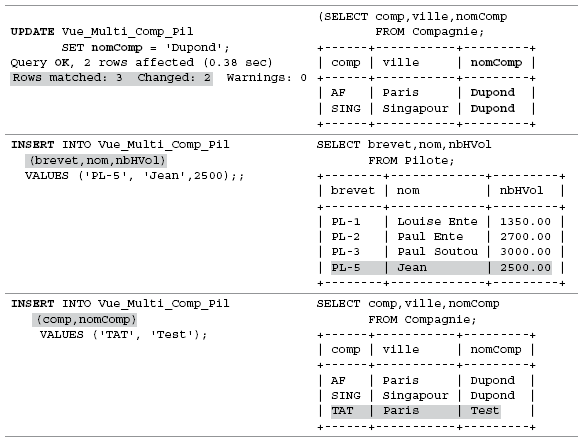
Aucune suppression n’est possible.

Les insertions sont permises seulement en isolant toutes les colonnes d’une seule table source.

Attention aux effets de bord quand vous modifiez une colonne provenant d’une table non protégée par clé. Il est plus naturel de modifier directement la table en question.

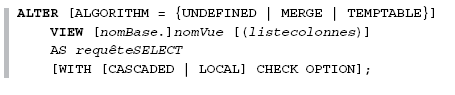
Modifions de différentes manières la vue multitable Vue\_Multi\_Comp\_Pil. La première tente une suppression, les deux suivantes modifient tantôt une colonne de la table protégée, tantôt une colonne de la table non protégée. Les deux dernières instructions insèrent dans chacune des deux tables.





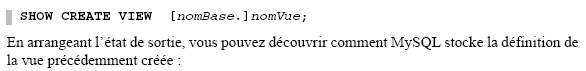
## Modification d’une vue (ALTER VIEW)

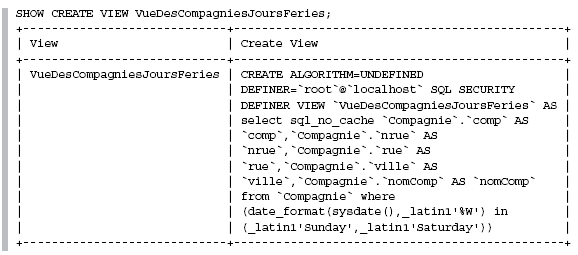
Vous devez au moins posséder les privilèges CREATE VIEW et DELETE au niveau d’une vue pour pouvoir la modifier. La syntaxe SQL est la suivante :



## Visualisation d’une vue (SHOW CREATE VIEW)

Pour pouvoir visualiser la requête de définition d’une vue, l’instruction que MySQL propose est la suivante :





## Suppression d’une vue (DROP VIEW)

Vous devez posséder le privilège DROP sur une vue pour pouvoir la supprimer.

La suppression d’une vue n’entraîne pas la destruction des données qui résident toujours dans les tables.

La syntaxe SQL est la suivante :

