#001

/\*

SQL cours - Introduction

========================

Structured Query Language

-------------------------

Bienvenue sur cette formation en MySQL.

Dans cette introduction, je vais vous présenter un petit peu ce qu'est ce langage SQL ? Pour ceux qui éventuellement ne le connaissent pas et détaillé un petit peu ce qui vous attend sur cet ensemble de vidéo sur ce cours et tout ce que vous allez pouvoir apprendre.

On parle de langage SQL non pas comme un langage dit de programmation, il ne s'agit pas ici de programmer, de créer des programmes qui pourront être exécutée mais un langage qui va permettre de réaliser des requêtes.

On parle donc de Structure Query Language, un langage de requêtes structurées. On va tout simplement faire des requêtes sur une base de données avec un ensemble d'informations qui seront organisés, ordonnés, de manière relationnelle. On en parlera un petit peu par la suite.

Conçu en 1974, normalisé en 1986

--------------------------------

Ce langage SQL a été conçu en 1974 mais comme tout langage , toutes technologies, à la base elle n'était pas forcément prévue pour être utilisée par le grand public ou par un ensemble de beaucoup d'utilisateurs.

Donc, il a fallu définir un ensemble de règles par la suite et ce langage a été normalisé en 1986 pour pouvoir ensuite être utilisé par plusieurs utilisateurs avec le même système de règles, la même syntaxe, les mêmes normes, ce genre de choses.

Donald D. Chamberlin / Raymond F. Boyce

---------------------------------------

Les créateurs, les concepteurs du langage SQL sont Donald Chamberlin et Raymond Boyce. Ce sont les 2 concepteurs du langage SQL qui à la base, travaillaient chez IBM. Une entreprise que vous connaissez, le nom vous dit sûrement quelque chose.

Voilà pour le petit historique très rapide de la SQL.

Encore une fois, c'est pareil, si vous voulez en savoir un peu plus, si vous voulez un peu plus de détails sur l'origine de la SQL, vous pourrez vous renseignez sur internet pour trouver beaucoup de choses intéressantes.

Utilisé dans les SGBDR(R) :

---------------------------

Alors le SQL, on va l'utiliser dans des SGBD(R).

C'est le diminutif pour désigner les systèmes de gestion de base de données et parfois on entend même parler de base de données relationnelle puisque les bases de données reposent ici sur un modèle relationnel. J'aurais l'occasion de vous détailler, vous expliquer tout au long de cette formation dans tous les cas.

Alors les SGBD, vous en connaissez très certainement.

Beaucoup parmi vous en ont déjà utilisé très certainement. Pour vous sortir quelques exemples ici :

\* On a par exemple le gros systèmes de gestion Oracle qui est suffisamment connu ;

\* On a MySQL qui est très répandue aussi ;

\* On a SQL Serveur de Microsoft ;

\* On a SQLite ;

\* On a PostgreSQL ;

\* Et on en a beaucoup beaucoup beaucoup d'autres.

il y a vraiment beaucoup, je vous ai donné en tout cas les principaux, ce qui me venait en tête mais il en existe évidemment beaucoup d'autres.

Ces systèmes de gestion de base de données vont permettre tout simplement d'utiliser le langage SQL pour pouvoir effectuer tout un tas de commandes, tout un tas de requêtes et gérer des bases de données.

Permet de :

===========

Les bases de données grosso modo vont permettre pas mal de choses et notamment ce langage MySQL lui puisque c'est ça qui m'intéresse dans cette formation.

\* Stocker des données (modèle relationnel)

------------------------------------------

Pour faire un résumé grosso modo, il va vous permettre de stocker des données suivant un modèle relationnel donc on parle du modèle relationnel puisque les données ne vont pas être stockées n'importe comment.

Il ne s'agit pas par exemple d'avoir un fichier où on aurait mis tout un tas d'informations dedans sans qu'elle soit classée, trièes, ordonnées.

Ici on va les classer sur un système de relations c'est-à-dire que chaque donnée aura des relations avec d'autres mais je vais vous l'expliquer, vous allez le comprendre via un schéma que je vous montrerais un petit peu plus tard.

\* Manipuler ces données (requêtes, procédures, ...)

---------------------------------------------------

Ca vous permettra également de manipuler ces données c'est-à-dire qu'une fois que les données stockées, on peut les manipuler, on peut effectuer des requêtes, on peut même carrément mettre en place des procédures pour effectuer des traitements sur ces données là.

On peut bien évidemment ajouter des nouvelles données, ont peut en supprimer, en modifier, et cetera.

Donc c'est tout ce système là qui concerne les manipulations des données sur ces bases là en SQL.

\* Peut être employé dans un langage de programmation

----------------------------------------------------

On va pouvoir être bien évidemment, c'est là aussi tout l'intérêt de cette formation, l'employer dans un langage de programmation. Donc tout ce que vous allez apprendre sur cette formation SQL et que vous aurez aqui pourra vous servir avec d'autres langages.

Si vous suivez la formation en PHP ou C++, python et cetera. Et si vous êtes amené à un moment donné à faire de la base de données, du SQL, et bien utilisez tout ce que vous avez appris ici pour vous en servir, l'employer via un langage de programmation. Ce sera tout utile, c'est pour ça que je fais cette formation là. Elle est indépendante. Vous pouvez très bien faire du SQL sans (pour ça qu'on dit qu'il peut être employé) 'programmation'.

Vous pouvez très bien faire du SQL directement sous forme de procéduraux, en exécution direct dans le SGBD. On peut même carrément faire des instructions via une interface de programmation qui serait directement par exemple présenté en interne.

Le choix d'avoir des instructions SQL employé dans un autre langage de programmation c'est une possibilité mais ce n'est pas une obligation. On n'est pas obligé d'utiliser un langage de programmation pour manipuler SQL.

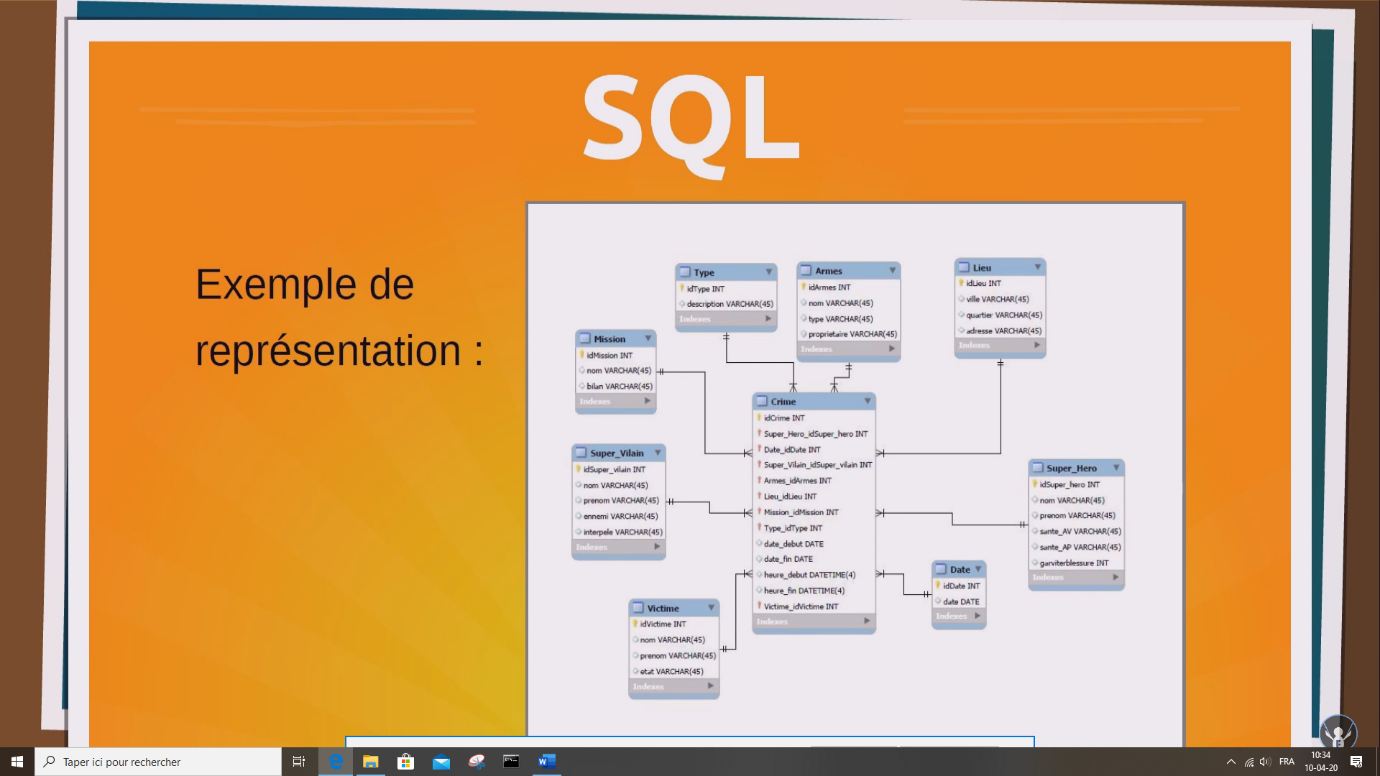
Les personnes par exemple qui gère des bases de données, ils ne font pas de programmation, ils ne font pas de PHP, ils ne font pas de C++, de python. Ils gèrent juste leur base de données via un système de gestion, terminé.

Mais vous dans ce cas, ça pourra vous servir éventuellement à vous en servir si vous le voulez dans un langage de programmation pour manipuler par exemple une base de données pour un blog en PHP ou en python, manipuler pareil un ensemble de données pour une application web ou SQLite éventuellement ou ce genre de choses.

Donc c'est pas mal et ça ne va pas mal être intéressant et surtout pourra vous servir pour votre formations ici.

Exemple de représentation

=========================



Voici un exemple de représentation d'une petite base de données parce qu'il faut savoir qu'une base de données peut représenter des milliers, des millions, des millions de données même plusieurs téraoctets de données sur certaines grosses bases de données par exemple vous pouvez prendre un hôpital, un CHU, vous pouvez prendre une gestion pour un magasin, pour tout un ensemble d'enseignes.

Tout cela va être géré dans des bases des données et puis imaginez bien qu'il peut y avoir des milliers, des millions de données différentes.

Donc là sur ce schéma, on se rend bien compte du système relationnel. C'est une base de données par exemple qui représenterait un système, qui représenterait une base de crime tout simplement, comme vous le voyez ici.

Et qui va posséder des relations, on voit par exemple qu'un crime peut être relié, donc il y a une relation avec un lieu. Il peut éventuellement avoir mis en évidence l'utilisation d'arme, ce genre de choses.

On va avoir un super vilain, on va avoir éventuellement un super héros, on aura une victime peut être potentiellement.

Tout cela va créer des relations. On voit que la table principal (c'est comme ça que ça s'appelle) et la table crime mais qu'elle peut être l'éventuelle reliées à d'autres tables.

Donc tout ces rectangles ce sont des tables. On apprendra à en créer, on apprendra à travailler dessus. Et toutes les informations à l'intérieur pour chaque table se sont les champs de la table. Et toutes ses liaisons que vous voyez là, ce sont en fait, les relations qui sont créés entre les différentes tables parce que ça n'aurait aucun intérêt dans une base de données d'avoir des tables qui sont chacune isolée. C'est pour ça qu'on parle de modèle relationnel il a forcément des relations avec les tables entre elles automatiquement. Si il a un crime, il ne peut pas rester isolé par rapport à un lieu, par rapport à une arme ou autre. C'est des choses qui sont reliés ensemble.

Donc voilà, un petit exemple de représentation que j'ai trouvé suffisamment explicité et bien illustré pour montrer un petit peu à quoi ça ressemble à quoi vous attendre au niveau de ce qui va être fait.

Par la suite on apprendra à faire pas mal de petites manipulations sur ce genre de base de données.

Notre choix se portera sur ...

==============================

Nous, pour cette formation, le choix se portera sur MySQL, pour la simple et bonne raison qu'il est très très répandu, il a une documentation qui est très fournie, une très grosse communauté aussi donc beaucoup de choses que vous pouvez trouver sur internet si un jour vous avez le moindre problème.

C'est une solution qui est gratuite, qui est open source, qui est multiplateforme donc voilà.

Disponible sur n'importe quel système d'exploitation donc ça paraît évidemment bien préférable contrairement par exemple à Oracle qui peut proposer une version payante ou SQLserveur de Microsoft.

Donc mon choix, s'est porté pour cette formation MySQL qui sera un bon choix pour ce que nous allons faire au sein de ces vidéos.

voilà pour ça.

Alors, je résume, grosso modo, un petit peu ce qui va nous attendre sur cette formation. La liste n'est pas exhaustive mais c'est histoire de vous faire un peu une idée de ce que vous allez trouver sur ce cours-là.

Dans ce cours, nous verrons comment :

=====================================

\* Installer le serveur MySQL et le démarrer

-------------------------------------------

Donc vous allez tout simplement voir comment installer le serveur MySQL et le démarrer.

Donc c'est ce que nous allons faire dans la vidéo prochaine.

On le fera pour Windows, pour Linux, j'en parlerai un tout petit peu pour Mac mais encore une fois je peux pas vous faire le détail pour MacOS puisque je n'ai pas de Mac sous la main mais je vous en parlerais un petit peu voilà pour que normalement vous puissiez vous débrouiller et installer aussi de votre côté.

On verra tout ça pour que vous ayez l'outil en place.

\* Exécuter nos premières commandes SQL

--------------------------------------

Donc on parle de serveurs, on va démarrer un serveur MySQL et ensuite on pourra éventuellement se connecter dessus et faire tout un tas de choses donc on verra ça par la suite.

Vous verrez comment éxécuter vos premières commandes MySQL donc de voir comment se passe la syntaxe ? comment ça s'écrit ? comment sera envoyé les résultats ? de quelle manière ? donc on travaillera un petit peu là dessus pour exécuter nos première commandes ?

\* Créer nos premère tables, organiser les relations

---------------------------------------------------

Une fois que vous aurez vu ça, on pourra commencer à voir pour créer une première table, organiser des relations entre celles-ci, organiser une information, qu'elle donnée on stocke ce genre de chose ...

Pour la suite, quand vous aurez bien appris ces chose-là, je pense que je vous proposerai un script SQL que vous pourrez importer chacun de votre côté sur votre serveur pour en fait travailler sur la même base de données.

On aura la même base de données, avec les même tables, les mêmes champs et cetera.

Pour toutes les vidéos que vous suivrez et bien on travaillera sur les mêmes données, ce sera beaucoup plus simple pour vous aider et pour que vous puissiez apprendre.

Le fait de travailler un petit peu tous sur les mêmes données, ce sera beaucoup plus pratique donc on fera ça.

Je vous proposerai ce script SQL en descriptions de vidéos, dans la séance pour laquelle ça sera utile de l'avoir.

Donc on verra tout ça.

\* Faire des requêtes, slectionner des données, en ajouter...

------------------------------------------------------------

Bien évidemment, une fois que vous serez un petit peu faire toutes ces choses-là, on commencera à travailler sur nos bases de données.

On fera des requêtes, on ira sélectionner des données, on ira en ajouter ou en supprimer, en modifier.

\* Tirer le plein portentiel du langage (optimisation !)

-------------------------------------------------------

Bref on ira faire tout un tas de de manipulation là dessus pour voir comment que ça marche avec SQL et de manière générale on verra comment tirer le plein potentiel (comme je l'ai marqué) du langage c'est-à-dire bien de veiller à faire des bonnes requêtes parce que vous verrez qu'en SQL, on peut avoir des requêtes tellement mal optimisé ou tellement mal écrites que leur exécution serait très longue voilà très très très longue, pas optimisé du tout et mangerait beaucoup de ressources sur vos machines.

Donc nous on verra sur vos machines pour faire des choses très optimisées, les plus rapide possible, les plus optimales dans la manière de concevoir les requêtes, dans les choses qui sont à faire ou plutôt à éviter.

Donc on verra tout ça tout au long de ce cours là pour que vous aillez les bonnes habitudes dés le départ et les bons réflexes pour pouvoir réaliser de très bonnes requêtes SQL.

Parce que les requêtes, on trouve vraiment à boire et à manger et n'importe quoi, des requêtes qui vont être très bien faites, très bien optimisées et d'autres qui devrait même pas exister.

Du coup ça rend les exécutions vraiment très très longue pour pas récupérer grand chose.

Privilégier l'optimisation pour pouvoir avoir des requêtes qui sont rapides et réactives.

A bientôt pour la suite !

Nous auront eu le départ au niveau de cette introduction pour ce qui nous attend au niveau de cette formation SQL.

C'est une formation qui servira dans beaucoup de choses. Elle nous servira juste pour manipuler des bases de données.

Encore une fois, elle nous servira avec nos langage de programmation que vous étudiez, que vous allez étudier ou que vous connaissez déjà pour manipulez nos bases de données.

Mais aussi pour travailler sur des applications web, en réseau, ce genre de choses...

A bientôt pour la séance 2 de MySQL.

\*/

#002

SQL cours - Installation mysql

Windows

=======

Installation manuellement un serveur via un client

--------------------------------------------------

Télécharger et installer l'archive de MySQL Community Server afin de paramétrer sans passer par un installateur.

Logiciel open-source sous licence GPL (GNU General Public License).

https://dev.mysql.com/downloads/mysql-8.0.19-winx64.zip

Dans le répertoire, je Cree un dossier MyWamp afin de dézipper mon dossier MySQL téléchargé. Je renomme ce dossier

mysql, il prendra place à côté de mon dossier Apache et PHP.

WAMP est un acronyme informatique signifiant : « Windows » « Apache » « MySQL ».

Créer un fichier de configuration et définir les chemins

--------------------------------------------------------

Créer un fichier de configuration que l'on appelle « my.ini » (nom par défaut Windows) dans C:\MyWamp\mysql.

Il permettra d'afficher les chemins.

Note : mysqld avec d pour driver de MySQL. Il peut être installer n'importe où dans le pc mais on l'installe

dans C:/ (sur Windows, on recommande de toujours mettre des slash). Les chemins peuvent être écrit avec des

antislashs mais dans ce cas, il faut en mettre 2, \\.

[mysqld]

basedir=C:/MyWamp/mysql

datadir=C:/MyWamp/mysql/data

[mysqld] = première ligne de notre fichier de configuration. Il est noté mysqld avec un d pour le driver du

serveur MySQL.

basedir = chemin du répertoire de base de MySQL. Il prend en compte les slashs ou deux anti-slashs.

Datadir = chemin vers le répertoire de base de données, les logs, etc. Il serra générer s'il n'existe pas.

Initialisation du répertoire Data

---------------------------------

On procède à l'initialisation du répertoire Data en se servant du fichier de configuration.

D'abord on ferme le fichier de configurations et on ouvre un invite de commande.

cd C:\MyWamp\mysql\bin

.\mysqld.exe --defaults-file=C:\MyWamp\mysql\my.ini --initialize --user=mysql

".\mysqld.exe" permet d'exécuter le driver auquel on lance l'initialisation.

"--defaults-file=<chemin>" : option qui permet d'écrire le chemin vers le fichier de configuration.

"--initialize" : option qui initialise le répertoire de données.

"--user" : option pour spécifier un utilisateur (automatique sur Linux). Non essentiel sur Windows.

On lance la commande et le répertoire de données 'data' va se créer dans mon dossier mysql.

Remarque : Par défaut, on a un super utilisateur root qui permet de se connecter via MySQL pour lequel un mot

de passe a été définis par défaut. => Ce mot de passe est simplement donné dans les logs.

Démarrer le serveur MySQL

-------------------------

mysqld.exe --console

Comme un serveur Web, Apache, une fenêtre s'ouvre et doit rester ouverte.

Tant que la fenêtre serra ouverte, le serveur restera actif. Ce sont des serveurs qui utilisent des sockets,

des TCP, etc.

Récupérer le mot de passe par défaut

------------------------------------

Le fichier C:/MyWAMP/mysql/data/\*\*\*\*.err (comme error) porte le nom réseau de notre ordinateur.

C'est un fichier de logs très important situé dans C:/MyWAMP/mysql/data. C'est un fichier log, il contient

notre mot de passe par défaut pour se connecter à notre serveur.

Il contient un mot de passe temporaire généré pour un utilisateur local root :

A temporary password is generated for root@localhost: \*\*\*\*\*\*\*\*\*\*\*

Remarque : Ici on ne créera pas de nouveau utilisateur.

Se connecter au serveur MySQL en tant que client

------------------------------------------------

Dans la mesure où le serveur MySQL est toujours lancé, ouvrir un autre terminal qui permet de se connecter en

tant que client au serveur mysql et donc d'accéder à nos bases de données.

Accéder aux exécutables de MySQL (dossier des binaires) et lancer notre exécutable.

cd C:\MyWamp\mysql\bin

mysql -u root -p

"-u" pour le nom d'utilisateur à utiliser (root).

"-p" car nous avons un mot de passe.

Remarque : On peut ajouter le chemin (comme pour MinGW pour la compilation en langage C sur Windows)

"C:\MyWamp\mysql\bin" à la variable d'environnement PATH.

Cette variable qui va être consulté par notre terminal quand on écrit une commande.

En ajoutant notre chemin anti-slash des binaires (bin) dans les variables d'environnements, il suffit de taper

mysql dans un terminal pour se connecter au serveur.

Entrer le mot de passe présent dans le fichier .err et commencer à utiliser les requêtes vers le serveur MySQL.

Pensez à bien terminer par un point-virgule.

Modifier le mot de passe root

-----------------------------

alter user 'root'@'localhost' identified by '';

"alter" : commande qui permet d'altérer/modifier une table (exemple alter user).

L'élément entre quottes à la fin remplace le mot de passe actuel (pas de mot de passe si un vide).

Relancer le terminal pour confirmer le changement de mot de passe :

cd C:\MyWAMP\mysql\bin

mysql.exe -u root -p

Utiliser des premières requêtes

-------------------------------

Note et pour aide : help ou \h ; clear ou \c.

SHOW DATABASES;

C'est une commande qui permet de lister, d'afficher les bases de données.

On voit ici s'il prend bien en compte le changement (Aucun password demandé). On relance la commande

Show database afin de voir la base de données par défaut pour la version 8.0.11 MySQL Community Server - GPL

(ne pas modifier).

mysql> show databases;

+--------------------+

| Database |

+--------------------+

| information\_schema |

| mysql |

| performance\_schema |

| sys |

+--------------------+

5 rows in set (0.03 sec)

Quitter proprement le serveur

-----------------------------

Ouvrir un nouveau terminal avec les commandes :

cd C:\MyWAMP\mysql\bin

mysqladmin.exe -u root -p shutdown

exit

Fermer le terminal qui avait démarré le serveur serveur avec la commande :

exit

En résumé,

----------

\* On a téléchargé l'archive et on la place là ou on veut (dans le C) ;

\* On a fait un fichier de configuration et on a initialisé les données via ce fichier pour avoir tous les

chemins qui correspondent ;

\* On a démarré notre serveur MySQL et on s'est connecté via le client au serveur ;

\* On a modifié le mot de passe root ;

\* On a fait une première requête MySQL qui nous affiche les bases de données

Linux

=====

L'installation est basée sur un système tel que Debian, Ubuntu, Elementary, Mint mais il faudra faire autrement

pour Fedora, Arch Linux. Ici MySQL va fonctionner comme un service équivalent aux pages de fond sur Windows. Il

ne nécessite pas qu'un terminal soit ouvert pour fonctionner.

Installer MySQL et mettre à jour le système

-------------------------------------------

Ouvrir un terminal et mettre à jour les dépôts :

sudo apt-get update

Ensuite,installer le paquet MySQL présent dans les dépôts de canonical pour Ubuntu :

sudo apt-get install mysql-server

Une fenêtre apparaît pour changer le mot de passe de l'utilisateur.

Si ce n'est pas le cas, ouvrir le fichier /var/log/mysql/error.log si le mot de passe de l'utilisateur root par

défaut existe.

cd var/log/mysql

gedit error.log

Commandes de bases pour gérer le service MySQL

----------------------------------------------

Le serveur sur le système Linux va démarrer comme un service, c’est-à-dire qu'il va tourner en tâche de fond

(comme les services sur Windows). Il faut alors stopper et quitter le service quand on a plus besoin. On pourrait

créer un service sur Windows mais en général c'est propre au système. On préfère généralement démarrer et éteindre

les systèmes en fonction des besoins plutôt qu'un service qui démarre au démarrage. On peut néanmoins se renseigner

sur comment créer un service sur Windows.

Démarre un service mysql en tâche de fond, je peux fermer mon terminal :

sudo service mysql start

sudo service mysql restart

Arrête le système :

sudo service mysql stop

Relance la configuration :

sudo service mysql reload

Force la rechargement de la configuration :

sudo service mysql force-reload

Tester la version installée de MySQL

------------------------------------

mysqld --version

Remarque : la version est généralement plus ancienne sur Ubuntu que sur Windows.

Vérifier le fichier de configuration dans le dossier ETC

--------------------------------------------------------

cd etc/mysql/conf.d/

ls

Le fichier "mysql.cnf" est autoconfiguré sur Linux. L'extension est .cnf et pas .ini comme on l'as installé

sur Windows. Sur Linux nous n'avons pas besoin de les ouvrir.

Se connecter à nouveau en tant que client sur le serveur MySQL

--------------------------------------------------------------

Requêtes pour se connecter en tant que client sur le server et afficher les bases de données :

sudo mysql -u root -p

Si nécessaire, on retire le mot de passe :

alter user 'root'@'localhost' identified by '';

Afficher les bases de données :

show databases;

On évite de faire tourner le système pour rien :

exit

sudo service mysql stop

rem. Créer un nouvel utilisateur (recommandé) :

https://stackoverflow.com/questions/39281594/error-1698-28000-access-denied-for-user-rootlocalhost

# Configuration encodage

SQL #3 - configuration encodage :

https://www.youtube.com/watch?v=bkrNOyo1gYk&list=

Afin de maitriser la configuration de notre environnement et les outils que nous allons utiliser.

Configurer l'encodage (la gestion des différents caractères) permet l'usage des chaînes de caractères. On communique dans différentes langues et notamment dans la langue anglaise qui ne possèdent pas de caractères accentués ou spéciaux. C'est la langue de base qui est le mieux géré par les systèmes informatiques. Tandis que le français (caractères accentués), l'allemand (caractères spécifiques), le japonais, le chinois, le thaïlandais, le Corréen (idéogrammes, kanji, etc.) pourrait poser des problèmes de lecture.

C'est pourquoi, on travaille de préférence avec une table de caractère Unicode pour tout nous fichiers. Au niveau de l'éditeur de texte, nous allons encoder tous nos fichiers en UTF-8. Ce type d'encodage va gérer toutes les langues du monde.

Toutes les applications (l'éditeur de texte, le serveur et les clients, fichiers .py, .html, etc) dans un domaine informatique doivent être configurer de la même manière en Unicode. Tous nos fichiers doivent être encoder en UTF-8 (idéalement tous nos fichiers doivent être convertit en UTF-8). En SQL, on se sert peu de l'éditeur sauf lorsqu'on écrit des scripts avec des requêtes. Sous Visual Studio, le type d'encodage se trouve en bas à droit, il suffit de cliquer dessus pour le configurer en UTF-8 une fois.

MySQL est un cas particulier de langage, on encode nos chaînes de caractères en UTF-8 mais codé sur 3 octets, ce qui ne permet pas de prendre en compte tous les types de caractères qui existent. Par exemple, il y a des caractères très spécifiques en Unicode, ils ne peuvent pas être formaté, pris en compte avec de l'UTF-8 simple. C'est pourquoi, on travaille sur un encodage de 4 octets (encodage qui varient entre 1 et 4 octets maximum).

Pour les autres langages, UTF-8 standard prend déjà tout en compte. C'est juste que pour MySQL, UTF-8 prenait que jusqu'à 3 octets à l'époque. C'est pourquoi ils ont rajouté un encodage spécifique pour pouvoir aller jusque 4 octets par d'autres caractères sont ajouté en Unicode, il faut pourvoir les stocker sur de plus grand espace en mémoire.

L'intérêt pour MySQL est donc d'être codé sur 4 octets et ne plus être limité à un encodage sur 3 octets.

## Windows

### On édite notre fichier de configuration my.ini (de base)

Nous rajoutons quelques lignes pour spécifier les encodages.

[mysqld]

basedir = C:/MyWAMP/mysql # Chemin vers la racine du répertoire MySQL (où sont enregistrés les bases de données).

datadir = C:/MyWAMP/mysql/data # Chemin vers le répertoire des données (où sont enregistrés les données).

On ajoute deux lignes pour le serveur dès le début de configuration pour éviter de convertir par la suite des tas de données.

*Attention : faire de la conversion d'encodage sur une base de données qui existe déjà n'est pas une bonne chose. Peu de chance que tout se passe bien dès le début. Devoir optimiser ou réparer une table n'est pas plaisant.*

character-set-server = utf8mb4

Ici on a la plage d'encodage pour le serveur lui-même. 'utf8' ou 'utf8mb3' qui prend maximum 3 octets.'utf8mb4' prend maximum 4 octets (4 \* 8 bits, 32 bits pour coder des chaînes), la plus grande plage disponible pour les chaînes de caractères. C'est l'encodage pour les chaînes de caractères, le serveur.

collation-server = utf8mb4\_unicode\_ci

Ici on a la plage d'encodage des chaînes de caractères pour les données tel qu'elles sont stockées (base de données, tables, etc.). ci (case insensitive) ou cs (case sensitive, sensible à la case) ou as ( accent sensitive).

Pour le CI, "matable", "MATABLE" et "maTABLE" ou 'e' et 'é' sont identiques.

[mysql]

default-character-set = utf8mb4 # l'encodage des charactères pour mysql.

[client]

default-character-set = utf8mb4

Ici, l'encodage des charactères à copier pour chaque nouveau client qui se connecterait. On peut copier/coller la ligne pour chaque nouveau client.

### Vérifier que tout est utf8mb4.

On vérifie que tout est en utf8mb4 avec my.ini (de base) et les commandes suivantes.

mysqld.exe --console # démarrer le serveur (dans un terminal à part).

mysql.exe -u root –p # démarrer MySQL.

On regarde que tout est au mieux encoder en utf8mb4 avec la commande show.

show variables like 'char%'

Ici, on affiche les variables commençant par char suivi d'un % (pour dire qu'il y a quelque chose après). On constate que les encodages par défaut ne sont pas toujours identiques (cp850, binary, utf8, utf8mb4), l'utf8mb4 est le plus efficace bien qu'intégré que pour certaines parties.

show variables like 'collation%'

exit # quitter MySQL.

mysqladmin.exe -u root -p shutdown # quitter le serveur.

*Remarque : Toutes les variables ne peuvent pas être changé (écrasement de valeur) en utf8mb4, notamment le système reste en utf8 et le filesystem en binary.*

*Attention : Quand on veut modifier la configuration de MySQL, il ne faut pas avoir le serveur qui tourne. On coupe le serveur, on modifie la configuration et on redémarre le serveur. Risque de complications.*

## Linux

### On édite ensuite mysqld.cnf

sudo gedit /etc/mysql/mysql.conf.d/mysqld.cnf

*Note : pensez à enregistrer le fichier d'origine en tant que backup pour pouvoir le restaurer si nécessaire.*

*Remarque : les WARNING \*\* ne pose pas de problème (erreur identique dans la vidéo).*

### On réécrit nos configurations à la suite du fichier.

[mysqld]

character-set-server = utf8mb4

collation-server = utf8mb4\_unicode\_ci

[mysql]

default-character-set = utf8mb4

[client]

default-character-set = utf8mb4

### On démarre ensuite le service pour s'assurer que tout est bon.

sudo service mysql start # démarrer le service.

sudo service mysql reload # recharger les configurations (Pour être sûr).

sudo mysql -u root -p # se connecter au serveur.

On regarde que tout est au mieux encoder en utf8mb4 avec la commande show.

show variables like 'char%'; # afficher les variables commençant par char suivi de quelque chose (%).

show variables like 'collation%';

*Remarque : au niveau de l'encodage pour le système de connexion, c'est à faire avec le système avec lequel on se connecte. C'est avec les langages PHP, ruby ou autres qu'on gérera la connexion à nos bases de données. Requête pour tout enregistrer et se connecter en utf8mb4 au niveau de la connexion … -> SET NAMES utf8mb4 COLLATE utf8mb4\_unicode\_ci ;*

### Arrêter le serveur.

exit

sudo service mysql stop

exit

# Moteur de stockage

SQL #4 - Moteur de stockage :

https://www.youtube.com/watch?v=bkrNOyo1gYk&list=PLrSOXFDHBtfGl66sXijiN8SU9YJaM\_EQg&index=4&t=0s

On aborde les moteurs de stockage en SQL (transactionnels et non transactionnels) pour connaître leurs spécificités et savoir dans quelle situation utiliser l'un ou l'autre.

Le langage n'est pas compliqué mais c'est bien plus compliqué d'avoir un langage optimisé.

Il faut travailler dés le début avec le bon moteur de stockage plutôt que d'attendre d'avoir une base de données de plusieurs giga-octets ou téraoctets et de devoir changer en cours de route. Il est préférable d'avoir le bon environnement pour nos bases de données plutôt que de faire les changements une fois qu'on a alimenter nos bases de pleins de données.

Le moteur de stockage c'est un moteur qui définit certains algorithmes de stockage et la manière dont on accède aux données par un ensemble de requêtes.

mysqld.exe --console # démarrer le serveur (dans un terminal à part).

mysql.exe -u root –p # démarrer MySQL.

## Les différents moteurs

Plusieurs types de moteur différents qui sont gérés par défaut suivant le type de base, la version de MySQL ou bien le type de SGBD. On définit ainsi deux types de moteur, des avantages et des inconvénients :

### Le moteur transactionnel

Ce sont des moteurs qui disposent de systèmes de sécurité. Pour un certain nombre de client, on va exécuter des transactions à l'aide de requêtes sur la base de données mais en cas de coupure de courant, le système va reprendre la dernière requête en cours et s'assurer que tout s'est bien passé (pas d'interruption ou données tronquées, etc). Si ce n'est pas le cas, on pourra revenir en arrière, faire un roll back (revenir à l'état de la base de données avant la requête qui n'as pas fonctionné). Un moteur dit transactionnel serra moins rapide car avec chaque requête s'accompagnera une surcouche qui va protéger l'état des requêtes et surtout de ma base de données par l'application de sécurité.

*Exemple de moteur de stockage transactionnel :* ***InnoDB*** *est plus sécurisé et favorise l'intégrité des données**(sécurité, intégrité des données, s'il y a beaucoup d'écriture, favoriser la cohérence entre les données (tables) càd que si les données ne sont pas utilisées, elles seront supprimées (exemple : plus aucun client sur Paris)).*

### Le moteur non transactionnel

Ce sont des moteurs plus performant et plus rapide parce qu'ils n'ont pas de système de sécurité. En cas de coupure, il n'y a pas de sécurité quant à la protection des données même si les requêtes seront plus rapides, moins gourmandes en mémoire. Par exemple, si je fais un SHOW DATABASES; il n'y a aucune sécurité que les données soient ou non corrompue en cas de coupure de courant. Si ça plante au milieu de la requête, tant pis seule la moitié de la requête aura été fictive sur l'ensemble des données et le reste non, d'où des corruptions, des erreurs, etc.

*Exemple de moteur de stockage non transactionnel :* ***MyISAM*** *est plus performant mais avec un risque d'avoir des problèmes avec les données**(peu gourmand en ressource, rapide en écriture, très rapide en lecture - récupérer des données de la base -, efficace pour la recherche). Choix si on a beaucoup de lecture/recherche à faire. Il assure la cohérence (relations correctes et cohérentes) des données plutôt que l'intégrité (données non corrompues, abimées, tronquées). Il bloque la table complétement s'il y a une requête d'écriture en cours, il mettra les autres utilisateurs en attente.*

### Memory (pas de place sur l'espace disque, tout en RAM)

C'est un moteur qui va travailler sur la mémoire vive. Contrairement à MyISAM, InnoDB qui stockent les données sur le disque dur de la machine, MEMORY les stocke dans la mémoire vive. Cela en fait le moteur de stockage le plus rapide de MySQL. Cependant, c’est aussi le plus dangereux puisque les données seront effacées de la mémoire vive après un simple redémarrage ou la moindre panne de la machine.

https://www.supinfo.com/articles/single/44-etude-plusieurs-moteurs-stockage-mysql

#### Avantage et application

Son avantage est qu'il permet de travailler sur des données temporaire, volatile quand on pas de place sur l'espace disque, tout est en RAM.

Il permet de travailler sur des données qu'on a besoin d'utiliser un court instant avant leur effacement.

## Choix du moteur de stockage

### Choix de la performance

En termes de performance, les requêtes sont plus rapides sur le moteur non transactionnel car il n'y a pas de système de sécurité. Ce moteur prend moins d'espace disque, donc moins gourmand en ressource sur notre serveur et sur notre ordinateur.

### Choisir un moteur ou les prendre tous les deux …

En termes d'utilisation, il ne faut pas forcément privilégier absolument un ou absolument l'autre. Par exemple sur un forum, l'utilisateur n'aura pas envie d'attendre 20 secondes pour que les messages s'affichent. La bonne habitude est de savoir a quel moment utiliser l'un ou l'autre. C'est pourquoi on peut créer certaines tables avec un tel moteur de stockage et d'autres avec tel autre moteur de stockage.

### … mais ne pas mélanger les moteurs de stockage

Quand on a des tables qui travaille ensemble. Il faut éviter de mélanger une table transactionnelle et une table non transactionnelle. Par exemple, si ma table de client est écrite avec InnoDB et que la table des villes est écrite avec MyISAM, ça ne sera pas efficace de faire des liens entre les deux.

### Éléments non exhaustifs pour identifier le type de moteur dont a besoin

#### MyISAM

🡪 Rapide en écriture mais surtout très rapide en lecture. C'est un moteur fait pour la lecture, plus que l'écriture, suppression, etc. Il n'a pas d'intérêt sur une base de données où l'on écrit énormément. Pour un site de news, on préfère MyISAM pour obtenir un listing des news. Dans ce cas, on veut que ce soit rapide et que ça ne dure pas 15 secondes pour juste retourner un listing. Si sur notre site web, on lit plus de données que ce que l'on écrit, on utilisera MyISAM pour la table en question (on ne parle pas de toutes la base de données). Les tables qui stockeront les news utiliseront MyISAM de manière optimale.

🡪 Efficace pour la recherche. Par exemple si on veut créer un moteur de recherche tel que Google, Yahoo, etc. MyISAM va être très rapide dans les lectures. En effet, MyISAM c'est de la recherche et donc de la lecture.

#### InnoDB

🡪 Pour la création d'un article, on préférera attendre un peu plus avec InnoDB pour être sûr que l'article soit bien publié sur un forum. Un utilisateur va préférer attendre 2 secondes sur un forum pour poster son article plutôt que d'attendre 0,3 secondes pour avoir un message incomplet qu'il faut effacer et recommencer. Dans ce cas, ce n'est pas grave si l'écriture n'est pas très rapide.

🡪 Sécuriser les données et conserver l'intégrité. On s'en sert pour s'assurer de bien enregistrer les données.

🡪 Cohérence entre les données (tables). Par exemple, on a deux tables, une table client et une table ville. Si dans la liste des villes ont à Paris et qu'il n'y a plus aucun client dans la ville de Paris. InnoDB supprimera la ville. => C'est ce que l'on appelle une suppression en cascade. On garde une cohérence dans le sens où plus personne n'est à Paris, on supprime la base de données.

### Gestion des files d'attente

MyISAM bloque la table complète quand on travaille dessus et InnoDB bloque seulement la ligne.

#### MyISAM

Avec MyISAM, une seule requête de modification à la fois. La table se bloque pour les autres utilisateurs dès qu'un utilisateur lance une requête d'enregistrement, d'ajout, etc. Le ralentissement peut être important sur un site web lorsqu'on utilise un forum => attendre qu'un autre utilisateur écrive son message.

#### InnoDB

Avec InnoDB, une seule ligne est bloquée. Lorsqu'on veut modifier la ville d'un des clients, ça va juste bloquer ce client précis et pas toute la table. Et si à coté, quelqu'un veut lister les autres clients, il pourra le faire sans problème.

## Le moteur par défaut

Sur les versions antérieures à la 5.5 de MySQL, le moteur par défaut était MyISAM et maintenant c'est InnoDB. C'est un choix arbitraire que l'on peut modifier.

L'intérêt est lorsqu'on crée une nouvelle table. Elle va utiliser le moteur par défaut de notre système plutôt que nous laisser renseigner le type de moteur à chaque fois.

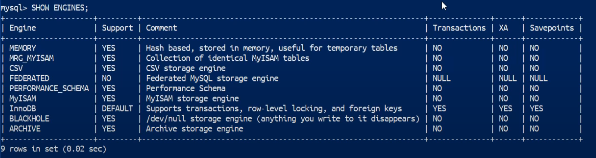
Il y a deux possibilités pour changer le moteur par défaut :

### Afficher les types de moteur

SHOW ENGINES; # Afficher les types de moteur sur MySQL.

La requête suivante affiche les moteurs disponibles, ceux qui sont supporter par défaut ou pas.

De cette manière, on voit que InnoDB est le moteur par défaut. Il est orienté sécurité parce qu'il gère les Transactions et les Points de sauvegarde. Les autres sont plus orienté performance.



### Changer le moteur par défaut

#### Soit via le client

Utiliser la requête SET puis un nom avec des underscores pour les commandes de configurations. Enfin, on affiche le moteur par défaut.

SET default\_storage\_engine = MyISAM; # changer le moteur par défaut.

SET default\_storage\_engine = InnoDB;

exit

mysqladmin.exe -u root -p shutdown

exit

#### Soit via le fichier de configuration

Modifier le moteur de façon permanente dans le fichier de configuration (my.ini sur Windows ou mysql.cnf sur Linux).

On édite la partie mysqld (c’est-à-dire la partie serveur) avec default-storage-engine composé de tirets. Les underscores sont utilisés avec le client.

[mysqld]

basedir = C:/MyWAMP/mysql

datadir = C:/MyWAMP/mysql/data

character-set-server = utf8mb4

collation-server = utf8mb4\_unicode\_ci

default-storage-engine = InnoDB

[mysql]

default-character-set = utf8mb4

[client]

default-character-set = utf8mb4

*Remarque : le paramètre est déjà par défaut à InnoDB, c'est pourquoi la ligne de configuration est facultative.*

## Sur Linux

Quand on modifie les configurations, il faut bien recharger la configuration et quitter le service à chaque fois sinon ce ne serra pas pris en compte.

sudo service mysql start # démarrer le service.

sudo service mysql reload # recharger les configurations (Pour être sûr).

sudo mysql -u root -p # se connecter au serveur.

...

# Gestion bases

SQL #5 - Gestion bases : https://www.youtube.com/watch?v=86f6ldnLVLM&list

Séance courte pour manipuler vos bases de données en SQL.

Les bases de données sont gérées sur le disque dur dans C:\MyWamp\mysql\data où sont stockés nos bases de données créées.

Une base de données est un fichier qui va stocker toutes les informations dans le répertoire data sauf si on travaille avec le moteur memory qui enregistre les données de manière temporaire sur la mémoire vive. Pour le cas de MyISAM et InnoDB, on va travailler directement avec des fichiers.

## Démarrer le serveur et se connecter en tant que client

mysqld.exe --console # démarrer le serveur (dans un terminal à part).

mysql.exe -u root –p # démarrer MySQL.

## Afficher les bases de données

SHOW databases; # Voir avant/après la création d'une base de données.

*Remarque : Prendre l'habitude d'écrire nos requêtes en majuscules sauf nos bases de données en miniscules.*

## Créer une base de données

CREATE DATABASE test; # Création d'une base de données en UTF8.

Ou bien pareil mais en apportant une petite sécurité en plus en demandant si la base de données existe. Sans le test, la console retournera un message d'erreur. En revanche avec le test, si la base de données existe, la console ne retournera rien.

CREATE DATABASE IF NOT EXISTS test; # Sécuriter afin de ne pas créer une base de données existante.

*Remarque : CREATE DATABASE ne pas partie des standards de SQL. Sur les autres SGBD, cette requête peut être renseignées d'autres options.*

## Vérifier l'encodage

SHOW VARIABLES LIKE 'char%';

*Remarque : database est en ut8mb4 si on a bien configuré notre encodage.*

## Demander à utiliser une base de données

Avant de travailler sur une base de données. Nous devons dire que nous l'utilisons.

USE test; # Utiliser une base de données.

## Supprimer une base de données

DROP DATABASE test; # Supprimer notre base de données.

## Eteindre le serveur

mysqladmin.exe -u root -p shutdown

exit

# Tables et données

SQL #6 - tables et données : <https://www.youtube.com/watch?v=Y_y-RNZApmk&list>

Dans cette vidéo, vous saurez créer/modifier/supprimer des tables SQL et connaître les types de données que vous pouvez utiliser pour stocker vos informations.

L'objet de ce chapitre est de voir comment intégrer des données à l'intérieur de notre base de données avec différents champs.

Une table est une sorte de boîte qui contient nos informations. Elle possède un nom et différents champs. Ces champs sont identifiés avec un nom, possèdent un type pour connaître le type de données stockées et possèdent également des contraintes (on parle de contraintes au niveau des champs ou des colonnes). La table peut être liées à d'autres tables SQL (on parle de SGBDr, relationnel). Il peut y avoir des tables liées entre elles ou isolées de tout l'ensemble de la base de données.

Pour commencer, on se connecte au serveur. On peut également créer un fichier code.sql (par exemple) pour écrire le code qu'on copie ensuite au niveau du client.

mysqld.exe --console # démarrer le serveur (dans un terminal à part).

mysql.exe -u root –p # démarrer MySQL.

## Écrire des commentaires en MySQL

-- Commentaire sur une seule ligne.

# Commentaire sur une seule ligne.

/\* Commentaire sur

plusieurs lignes \*/

## Requêtes sur les bases de données.

### Afficher nos bases de données

Par défaut des BDs sont déjà présentes. Sur un autre SGBD, d'autres peuvent s'afficher.

SHOW DATABASES; # Voir avant/après la création d'une base de données.

### Créer une base de données (et sous conditions)

CREATE DATABASE test; # Création d'une base de données en UTF8.

CREATE DATABASE IF NOT EXISTS `test` ; /\* Créer une base de données si elle n'existe pas déjà. Utiliser des anti-quote pour éviter de détecter un mot cle. \*/

Remarque : quand on fait appel au nom d'une base de données, au nom d'une table ou de l'un de ces champs. Il est recommandé d'utiliser des anti-quotes (AltGr + 7). En cas de nommage de notre base de données par le mot database, le serveur risque de le considérer comme un mot clé (mot réservé en SQL au niveau de la requête). Cela se voit avec la coloration syntaxique mais l'éditeur ne sait pas si c'est un nommage.

### Utiliser ou changer de base de données

USE `test`; # J'utilise cette base de données.

## Requêtes liées aux tables.

### Afficher toutes les tables de données depuis la BDD 'test'

SHOW TABLES;

La requête retourne *Empty set* si la table `test` est notre table courante. Sans la requête USE préalablement, la requête devient :

SHOW TABLES FROM `test`;

### Créer une table MySQL

Ce sont toujours les mêmes termes avec parfois quelques variations suivant les bases de données. Pour le SQL standard (commandes qui fonctionnent partout), les requêtes sont très faciles à retenir.

On crée une table `t\_users`. Le nom ne doit pas contenir d'accent, pas d'espace. Par exemple, en CamelCase (majuscule en chaque début de mot). Selon les systèmes, les noms sont sensibles à la case. Par exemple, `t\_users` est différent de `t\_USERS` suivant le système.

On peut également tester si la table n'existe pas déjà pour prévenir d'éventuel erreurs avec IF NOT EXISTS qui est facultatif.

CREATE TABLE `t\_users`

(

`id\_user` INT,

`user\_name` VARCHAR(65),

`user\_birthday` DATE

);

La requête se termine par des parenthèses où l'on peut enregistrer les différents champs et enfin terminer par un point-virgule.

#### Nommage recommandé pour les champs d'une table :

Il est recommandé de choisir un préfixe à une table (nom de famille, nom de site, initial, etc.), un underscore suivi du nom de la table qui dit à quoi cela correspond.

Par exemple, pour l'identifiant (id\_) qui est très important pour les index, les compatges, etc. il est recommandé d'indiquer id\_nomDeLaTable avec le nom au singulier. Et pour les autres champs, on indique inversement le nom de la table suivi du nom du champ (nomDeLaTable\_nomDuChamp).

Le nommage est important pour éviter par la suite la confusion d'un attribut simplement nommé `name` ou `id` dans différentes tables lors de requêtes de lecture.

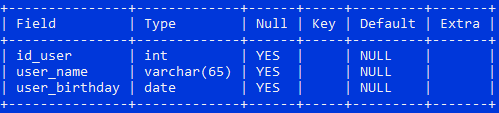
Ensuite, une fois le système et certaines règles, conventions choisies, il faut les garder jusqu'au bout afin de ne pas mélanger différentes syntaxes et que cela reste homogène.

### Afficher les détails d'une table

#### Détails d'une table (DESCRIBE)

DESCRIBE `t\_users` ;

Affiche les 3 champs de la table Users. EXPLAIN / DESC (diminutif de describe) sont des Alias qui peuvent remplacer DESCRIBE.



On voit les différents champs et types. Null, Key, Default, Extra concerne les contraintes.

#### Statut d'une table (SHOW TABLE STATUS)

Informations supplémentaires et plus détaillées que la commande avec DESCRIBE mais on s'en sert très peu. Des informations pour l'ensemble des tables comme le nom, le moteur de stockage, le nombre de lignes, la création, l'encodage.

SHOW TABLE STATUS ;

### Choisir un type de données optimal

Il faut bien choisir les types selon l'information à stocker. Le principe est de toujours choisir le type le plus limitant possible. Pour le stockage du nombre 99, le type de champ doit accepter un échantillon le plus petit possible pour le contenir. Il faut que le type soit suffisamment léger et optimiser pour réduire le nombre d'octet en mémoire.

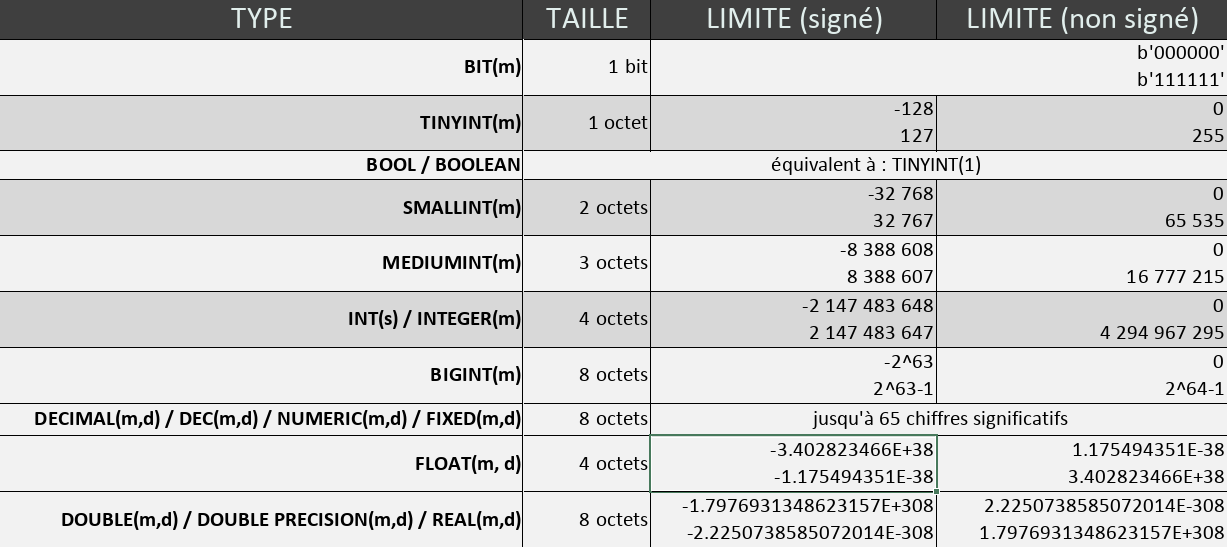
La règle est de se servir de ce qui est numérique pour le calcul et d'utiliser les chaînes de caractères pour tout ce qui ne concerne pas de calcul (codes postaux, l'âge mais dépend des cas). Cependant parfois, en termes de taille, une chaîne de caractères peut prendre plus d'espace en mémoire que si elle était stockée avec un type numérique.

On privilégie d'abord l'espace en mémoire avant le type de donnée. Ensuite on regarde si l'information peut doit servir dans un calcul ou non.

Certains des types de données présent sont valable uniquement que pour MySQL (BIGINT, VARCHAR, etc.). Le type INT sur SQL server ne sera pas le même type INT de MySQL.

Les tableaux suivants ne sont valables que pour MySQL.

#### Type concernant les données numériques (valable sur MySQL)



Pour rappel, on parle de nombre signé quand ils peuvent être positif ou négatif. En mathématique, ce sont des nombres arithmétiques. Un nombre qui n'est pas arithmétique ne peut être que positif. En informatique, on parle de nombre non signé. C’est-à-dire une valeur positive, un peu la valeur absolue en mathématique.

On a le binaire, le TINYINT pour le stockage d'un nombre comme 99 qui peut aller jusqu'à 255 en non signé. Pas besoin de plus d'un octet en mémoire pour le stocker.

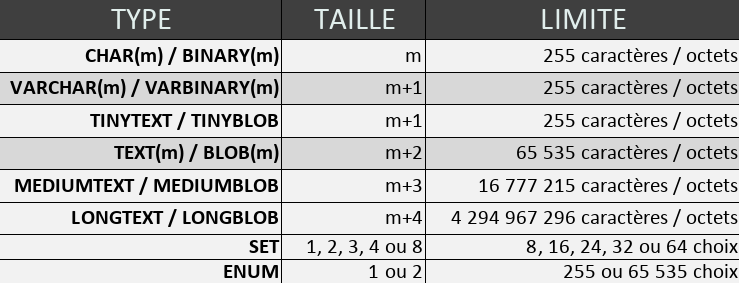
BIT(m) correspond à un TINYINT(1) de 1 avec m qui correspond à un paramètre. C’est-à-dire que si le paramètre est à 1, le nombre sera composé que d'un chiffre. Avec 2 en paramètre, on peut aller jusqu'à 99 et si 3 en paramètre, la limite serra fixée à 999 suivant la limite fixé (127 pour TINYINT). Autrement MySQL définit une taille par défaut qui n'est pas forcément la plus optimal.

Par exemple, un identifiant d'utilisateur `id\_user` est un numéro unique d'utilisateur qui démarrera de 1 pour atteindre le nombre d'utilisateur de notre site. Pour commencer, on ne s'entend pas à avoir 15.000 utilisateurs pour commencer, on peut ainsi s'orienter vers le type SMALLINT(m) pour commencer. Ce qui signifie qu'on aura beaucoup d'espace utilisé en mémoire que le fameux INTEGER(m). On réduit l'espace en mémoire de 4 à 2 octets, ce qui ne représente pas grand-chose mais une table de données peut faire plusieurs Giga, plusieurs Téra octets d'espaces (millions d'enregistrements). Dans ce cadre, le moindre octet de gagné est important.

Ensuite, le jour où l'on atteint les 65.000 utilisateur. Il suffira de faire une modification (ALTER) vers MEDIUMINT(m).

De cette manière, on prend l'habitude de toujours choisir le type plus efficace suivant l'information.

#### Types concernant les chaînes de caractères (valable sur MySQL)



Les chaînes de caractères sont utilisées lorsqu'il n'y pas de calcul à effectuer. Par exemple, les codes postaux sont des nombres sur lequel on n'a pas besoin de faire de calcul.

Chaque type a un équivalent en mode texte et un équivalent en mode binaire (c’est-à-dire stocké sous forme d'octets). Avec BINARY se sera stocké sous forme d'octets. Chaque homologue dispose de la même taille de stockage. C'est juste la manière dont est stocké les données qui changent.

Le paramètre m est fonction du nombre de caractères que nous avons.

SET et ENUM sont deux types particuliers qui donnent la possibilité de stocker une liste d'information. Par exemple, avec ENUM peut servir si on veut savoir si un utilisateur est abonné avec le champ `user\_is\_subscriber` ENUM(`Y`, `N`). Ce sera soit Oui, soit Non.

CREATE TABLE `t\_students`

(

`id\_user` SMALLINT,

`user\_name` VARCHAR(65),

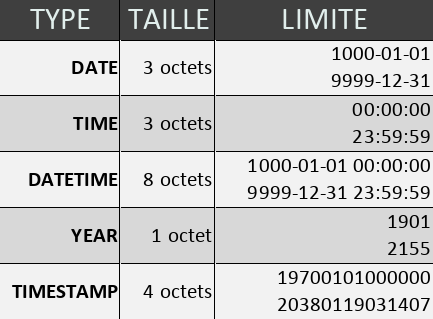
`user\_birthday` DATE,

`user\_is\_subscriber` ENUM('Y', 'N')

) ;

Pour le SET, c'est du choix multiple. On pourra choisir plusieurs valeurs. Tandis que ENUM, c'est un seul parmi la liste de 65 535 choix au maximum.

#### Types concernant les dates (valable sur MySQL)



En haut la limite inférieure et la limite supérieur en bas.

Le TIMESTAMP date de l'époque Unix. C'est le temps qui s'est écoulé depuis le 01 janvier 1970, depuis l'arrivée de la famille Unix. Depuis cette date-là un compteur calcule toutes les secondes écoulées. Cela permet notamment de connaitre le temps écoulé entre deux moments. Il s'agit donc d'un indicateur universel. Il est disponible partout (Web, Base de données, programmation, etc).

Encore une fois, on ne va pas prendre un DATETIME si le un champ DATE suffit. En effet, on passe de 3 octets à 8 octets.

### Altérer une table (Changer le type d'une variable déjà créée)

C'est seulement après avoir écrit ALTER TABLE qu'on peut ajouter des sous-requêtes de modification. On peut en ajouter plusieurs à la ligne et terminer avec un point-virgule à la fin.

ALTER TABLE `t\_users`

ADD `user\_points` TINYINT(2),

MODIFY `id\_user` SMALLINT,

CHANGE `user\_birthday` `id\_birth` DATE;

Certaines syntaxes sont valides sur tous les systèmes de gestion et d'autres non. Les requêtes ci-dessous sont généralistes

#### Ajouter une colonne (ADD)

Ajouter une colonne de TINYINT(2) de 'user\_points' TINYINT(2) dont la valeur max est 99.

ALTER TABLE `t\_users`

ADD `user\_points` TINYINT(2);

#### Supprimer une colonne (DROP)

ALTER TABLE `t\_users`

DROP `user\_points`;

#### Modifier une colonne (MODIFY)

Modifier une colonne et spécifier des informations (nouveau type, nouvelles contraintes, etc).

ALTER TABLE `t\_users`

MODIFY `id\_user` SMALLINT;

#### Changer le nom (et éventuellement le type) (CHANGE)

Change le nom d'une colonne et son type. CHANGE `ancienNom` `nouveauNom` <nouveauType>.

ALTER TABLE `t\_users`

CHANGE `user\_birthday` `id\_birth` DATE;

### Supprimer les relations de la table et supprimer une table (DROP)

#### Supprimer la table

Avant de supprimer une table, il faut d'abord supprimer ses relations.

DROP TABLE IF EXISTS `t\_users`;

Il est recommandé de tester la table si elle existe avant de la supprimer. On peut également afficher leurs informations.

SHOW TABLE STATUS ; # Informations supplémentaires à DESCRIBE.

SHOW TABLES ; # Vérifier les tables de données.

#### Supprimer la base de données

Changer la base de données et effacer la base de données en question.

USE mysql ;

DROP DATABASE test ;

Vérifier la suppression de notre base de données.

SHOW DATABASES ;

# Les contraintes

SQL #7 - contraintes : <https://www.youtube.com/watch?v=b3kbXSOg9ZU>

Les contraintes sont un ensemble de règles que l'on va définir sur les colonnes (champs) des tables SQL. Ce seront des règles à respecter lors de l'enregistrement d'informations.

Les contraintes doivent être définies dès le début pour éviter les problèmes et gagner en performance et cohérence entre les données.

## Rappel

Précédemment, on a vu cette syntaxe.

### Démarrer et se connecter.

mysqld.exe --console # démarrer le serveur (dans un terminal à part).

mysql.exe -u root –p # se connecter en tant que client.

### Créer une base de données

CREATE DATABASE IF NOT EXISTS `cours\_sql`;

### Utiliser la base de données

USE `cours\_sql`; # utiliser la base de données.

## Créer une table

On veut créer une table "users" avec un identifiant qui est un petit nombre entier, un pseudo qui a maximum 30 caractères (moins de caractères vaut moins d'espaces en mémoire), un é-mail qui peut occuper jusqu'à 100 charactères, une date d'enregistrement ou d'inscription et un booléen pour déterminer si l'utilisateur sera un admin ou non (1, admin).

CREATE TABLE IF NOT EXISTS `table\_users`

(

`id\_user` SMALLINT(6),

`user\_name` VARCHAR(30),

`user\_email` VARCHAR(100),

`user\_registration` DATE,

`user\_admin` BOOLEAN

);

## Colonne obligatoirement renseignée (NOT NULL : colonne obligatoire)

### Création d'une table avec la contrainte NOT NULL sur les colonnes

Les contraintes sont des règles qui doivent être logique, cohérente et du sens par rapport à l'enregistrement des données. On peut prendre l'exemple des formulaires HTML où l'on veut renseigner certains champs obligatoires (formulaire d'inscription, etc) avec à côté des informations dites facultatives.

La colonne de la table doit être obligatoirement renseignée quand on fait l'enregistrement d'un nouvel utilisateur. Par exemple, il suffit de renseigner tous les champs de la table avec NOT NULL sauf pour "user\_admin" où ce n'est pas à l'utilisateur de le renseigner. Souvent la date d'enregistrement est récupérée au moment de l'inscription avec un langage de programmation. Dans ce cas, même si ce n'est pas enregistré par l'utilisateur, on veut savoir à l'avance qu'une date a été obligatoirement renseigné pour chaque utilisateur.

Au moment de l'enregistrement, si l'utilisateur n'as renseigné la date, il y aura une erreur MySQL pour dire qu'on n'a pas respecter cette contrainte. Sans contrainte, le champ "user\_name" de cet utilisateur serait NULL.

Il suffit d'ajouter NOT NULL, écrite en minuscule ou en majuscule, avant la virgule.

CREATE TABLE IF NOT EXISTS `table\_users`

(

`id\_user` SMALLINT(6) NOT NULL,

`user\_name` VARCHAR(30),

`user\_email` VARCHAR(100) NOT NULL,

`user\_registration` DATE NOT NULL,

`user\_admin` BOOLEAN

);

### Procéder à une modification de la valeur

Procéder à un changement, une modification ou un ajout (ALTER). Ici on ajoute une contrainte NOT NULL au niveau d'un champ que l'on peut avoir oublié. Avec la requête suivante, on connaît les champs qui acceptent la valeur NULL.

DESCRIBE table\_users;

Procéder ensuite à la modification d'un des champs.

ALTER TABLE `table\_users`

MODIFY `user\_name` VARCHAR(30) NOT NULL ;

Toute la requête peut également tenir sur une seule ligne.

## Valeur par défaut numérique ou chaine de caractères (DEFAULT : valeur par défaut)

Par défaut, on veut que l'utilisateur ne soit pas désigné comme un administrateur. Par défaut, on renseigne la valeur 0 à "user\_admin". Si on veut passer quelqu'un en administrateur, on fait une requête pour renseigner la valeur 1 à son enregistrement.

Il suffit de passer DEFAULT au champ en question et de définir la valeur numérique ou en chaîne de caractères (DEFAULT "").

Enregistrer un champ par défaut sous-entend que le champ est non obligatoire. Il est rare qu'un champ obligatoire soit complété par une valeur par défaut. La contrainte DEFAULT permet quand même de remplir le champ.

### Création de la table

CREATE TABLE IF NOT EXISTS `table\_users`

(

`id\_user` SMALLINT(6) NOT NULL,

`user\_name` VARCHAR(30),

`user\_email` VARCHAR(100) NOT NULL,

`user\_registration` DATE NOT NULL,

`user\_admin` BOOLEAN DEFAULT 0

);

### Altération de la table

On modifie notre table (le champ en question) et on ajoute cette contrainte DEFAULT. On évite de faire des requêtes pour rien si le champ a déjà la contrainte.

Pour ajouter la contrainte si elle n'est pas déjà présente.

ALTER TABLE `table\_users`

ALTER `user\_admin` SET DEFAULT 0;

### Supprimer la contrainte

Il suffit de renseigner le nom de la colonne.

ALTER `user\_admin` DROP DEFAULT;

## Vérification d'une valeur sinon erreur mysql (CHECK : vérification d'une valeur)

### Création de la table

Par exemple, on ne veut pas que "id\_user" soit supérieur à 10.000. On peut aussi imaginer mettre la contrainte pour ne pas avoir un prix négatif (prix > 0).

Il suffit de renseigner le mot CHECK suivi d'un test à faire et si la contrainte n'est pas respectée, une erreur mysql.

CREATE TABLE IF NOT EXISTS `table\_users`

(

`id\_user` SMALLINT(6) NOT NULL CHECK(id\_user <= 10000),

`user\_name` VARCHAR(30),

`user\_email` VARCHAR(100) NOT NULL,

`user\_registration` DATE NOT NULL,

`user\_admin` BOOLEAN DEFAULT 0;

);

### Altération de la table

*Remarque : Parmi les contraintes à MySQL, certaines fonctionnent sur tous les systèmes et d'autres qui sont propre à MySQL, propre à Oracle, etc. Ainsi celles qu'on apprend ici sont standard.*

ALTER TABLE `table\_users`

ADD CHECK(id\_user <= 10000);

*Remarque : A ce stade, il n'est pas possible de vérifier. On peut afficher plusieurs informations avec les requêtes suivantes :*

SHOW TABLE STATUS FROM cours\_sql;

EXPLAIN table\_users id\_user;

### Vérification sur plusieurs champs (CONSTRAINT "nom")

Pour faire une contraintes CHECK sur plusieurs champs différents, il faut nommer la contrainte. Par habitude, on ajoute "chk\_nomDeLaContraindre" pour checker plusieurs éléments. C'est un moyen de faire des vérifications sur plusieurs champs.

ALTER TABLE `table\_users`

ADD CONSTRAINT chk\_idTooBig CHECK(`id\_user` <= 10000 AND `id\_user` > 0) ;

### Supprimer la contrainte

ALTER TABLE `table\_users`

DROP CHECK `chk\_idTooBig` ;

Remarque : il est recommandé d'avoir toutes les bonnes contraintes dés le début de la création de la table. Si un jour, on fait un oubli on peut utiliser ALTER TABLE mais c'est mieux d'avoir bien réfléchi dés le départ à avoir des bases de données cohérentes.

## Définir un champ de valeur unique (UNIQUE : valeur de champ unique)

Par exemple sur un site, le nom d'utilisateur est unique. Il ne faut pas que deux utilisateurs aient le même pseudo et se connecte sur le même compte. Risque de problème d'authentification et de vie privée. Ici "user\_name" devient une information unique qui génère une erreur sql en cas de répétition de pseudo à l'inscription.

Avant la création, il suffit d'ajouter UNIQUE() et de lui passer entre parenthèse, la valeur qui sera unique. On applique ainsi une contrainte d'unicité sur "user\_name".

### Création de la table

CREATE TABLE IF NOT EXISTS `table\_users`

(

`id\_user` SMALLINT(6) NOT NULL CHECK(id\_user <= 10000),

`user\_name` VARCHAR(30),

`user\_email` VARCHAR(100) NOT NULL,

`user\_registration` DATE NOT NULL,

`user\_admin` BOOLEAN DEFAULT 0,

UNIQUE(`user\_name`)

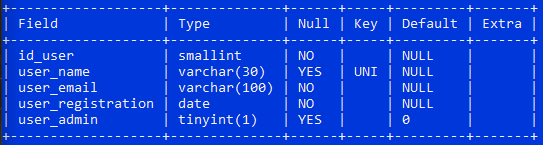
);

### Altération de la table

ALTER TABLE `table\_users`

ADD UNIQUE(`user\_name`);

MySQL génère une régle d'unicité qu'on observe avec la commande DESCRIBE. La colonne Key affiche UNI (PRI dans la vidéo pour PRIMERY KEY) pour le champ en question.



A partir de là, il ne peut plus y avoir d'utilisateur qui porte le même "user\_name" grâce à cette contrainte d'unicité.

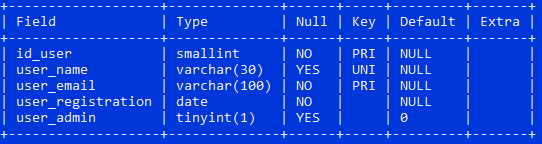
### Définir une contrainte d'unicité sur plusieurs champs

ALTER TABLE `table\_users`

ADD CONSTRAINT uc\_idUser UNIQUE(`id\_user`,`user\_email`);

La contrainte d'unicité se fait sur deux colonnes. Voici ce que l'on peut observer avec la commande.

DESCRIBE table\_users;



### Supprimer la contrainte

ALTER TABLE `table\_users`

DROP INDEX `uc\_idUser` ;

Seule la contrainte nommée est supprimée.

## Définir un élément unique (PRIMARY KEY : définir un élément unique)

Utilisé sur les identifiants (id\_user de la table) pour signifier un identifiant unique.

Différent d'UNIQUE car on ne peut avoir qu'une seule clé primaire mais plusieurs valeurs d'unicité.

### Création de la table

CREATE TABLE IF NOT EXISTS `table\_users`

(

`id\_user` SMALLINT(6) NOT NULL,

`user\_name` VARCHAR(30) NOT NULL,

`user\_email` VARCHAR(100) NOT NULL,

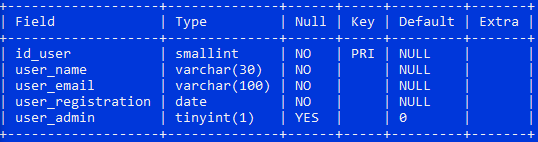
`user\_registration` DATE NOT NULL,

`user\_admin` BOOLEAN DEFAULT 0,

PRIMARY KEY(`id\_user`)

);

Nous obtenons une clé primaire sur "id\_users".



L'utilisateur n'aura pas à renseigner un identifiant au moment de l'inscription. C'est un identifiant qui va s'auto-incrémenter par programmation, un formulaire HTML, etc. SQL peut ajouter un identifiant jusqu'au maximum que peut accepter un SMALLINT(6). Cet identifiant est également renseigné à chaque fois (NOT NULL).

Il est recommandé d'appliquer également une autre contrainte (AUTO\_INCREMENT).

### Altération de la table

ALTER TABLE `table\_users`

ADD PRIMARY KEY(`id\_user`) ;

On ne pourra ajouter qu'une clé primaire mais plusieurs valeurs d'unicité car c'est définir qu'un champ est unique, on ne peut pas avoir des valeurs similaires. Cependant, vu qu'une clé primaire sert pour l'automatisme, on ne peut en avoir qu'une seule.

### Définir une clé primaire nommée c’est-à-dire sur plusieurs champs

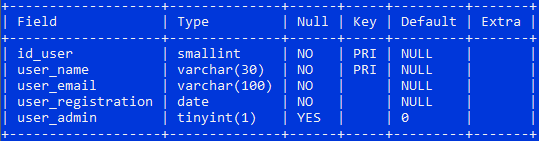
Par exemple définir une clé primaire sur les 3 premières lettres "user\_name" et les 3 premières lettres de l'émail.

Notre clé primaire est ici composée de "id\_user" et de "user\_name".

ALTER TABLE `table\_users`

ADD CONSTRAINT pk\_ident PRIMARY KEY(`id\_user`, `user\_name`) ;

Nous obtenons une seule clé primaire sur deux colonnes de notre base de données. Une seule clé primaire basé sur "id\_user" et "user\_name" à la fois.



### Supprimer la contrainte

ALTER TABLE `table\_users`

DROP PRIMARY KEY ;

## Incrémentation de 1 en 1 (AUTO\_INCREMENT  : incrémentation de 1 en 1)

### Création d'une table

CREATE TABLE IF NOT EXISTS `table\_users`

(

`id\_user` SMALLINT(6) NOT NULL AUTO\_INCREMENT,

`user\_name` VARCHAR(30) NOT NULL,

`user\_email` VARCHAR(100) NOT NULL,

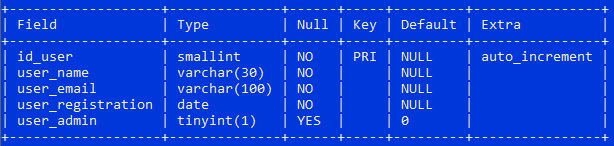
`user\_registration` DATE NOT NULL,

`user\_admin` BOOLEAN DEFAULT 0,

PRIMARY KEY(`id\_user`)

);

L'auto-incrémentation signifie que lorsque l'on enregistre le premier utilisateur, SQL va mettre 1 jusqu'à la limite de la taille de la donnée définie.



L'auto-incrément est bien renseigné à chaque fois dans Extra.

### Altération de la table

Pour changer l'endroit où ça s'incrémente.

ALTER TABLE `table\_users`

AUTO\_INCREMENT = 100 ; # incrément de 1 en 1 à partir de 100.

On choisit une incrémentation à partir de 100. Le premier utilisateur commencera à l'id 100. Par exemple pour un numéro de facture, on commence parfois à 1000, 10.000, etc.

Il est recommandé quand on crée un identifiant, on crée sur ce dernier, une clé primaire et une auto-incrémentation.

L'intérêt viendra lorsqu'on travaille avec des requêtes de sélection sur ces identifiants (et éventuellement d'un index), pour faire des parcours et qui seront plus rapide et plus performant en termes de ressources, etc. C'est ce que l'on fait en SQL pour créer des champs auto-généré (automatiquement par MySQL), on les met en NOT NULL, on les met en PRIMARY KEY (C'est comme un UNIQUE mais en plus fort) et AUTO\_INCREMENT.

Cependant, pour se créer un identifiant spécial, on prend les 3 premières lettres du nom de la personne suivi d'autre chose. On crée ensuite un identifiant unique avec la contrainte UNIQUE avec éventuellement un CHECK pour vérifier la valeur. Dans ce cas, l'identifiant est généré par un utilisateur.

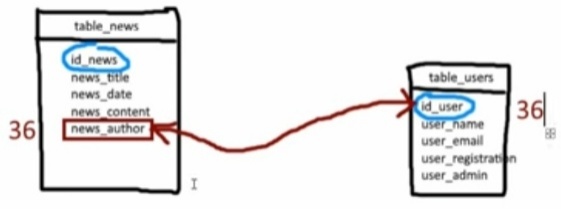
Ce cas est très important car il produits les relations entre les différentes tables. C'est pourquoi, on parle de système relationnel.

## Clé étrangère (FOREIGN KEY)

clé étrangère : C'est créer un lien entre 2 tables qui ont une colonne d'identifiant en commun. Le champ d'une table A est reliée au champ d'une table B. Il faut utiliser le moteur de stockage InnoDB avec la table qui utilise ces clés étrangères. Ça ne marchera pas avec MyISA.

### Relation entre les tables (Exemple de la gestion des news)

Dans l'exemple de la table des news (table\_news), on ne va pas renseigner tout ce qui se trouve dans la table des utilisateurs (table\_users) pour éviter les répétitions, les doublons et la redondance de données.



L'identifiant le plus fort de la news id\_news et l'identifiant le plus fort de l'utilisateur est également id\_user. L'utilisateur inscrit aura un id de 1, 2, 3, etc. De même pour les news auront des id de 1, puis 2, puis 3.

De cette information, on définit une relation entre news\_author qui sera l'identifiant id\_user relatif à la table des utilisateurs. Par exemple, un news\_author 36 correspondra à un id\_user de 36. Quand on fait requête pour faire des jointures, on est donc capable de relier plusieurs tables pour par exemple lister toutes les news faites par l'utilisateur 36 (la requête pourrait être faites sur user\_name/news\_author). A partir de ce système relationnel sur SQL, on peut retrouver toutes les informations de l'utilisateur en question sur base de l'identifiant. L'avantage c'est que l'on stocke juste un chiffre dans news.

On vient de définir une clé étrangère. **news\_author est une clé étrangère qui fait référence à id\_user** de table\_users.`

### Création d'une table

CREATE TABLE IF NOT EXISTS `table\_users`

(

`id\_user` SMALLINT(6) NOT NULL AUTO\_INCREMENT,

`user\_name` VARCHAR(30) NOT NULL,

`user\_email` VARCHAR(100) NOT NULL,

`user\_registration` DATE NOT NULL,

`user\_admin` BOOLEAN DEFAULT 0,

PRIMARY KEY(`id\_user`)

);

CREATE TABLE IF NOT EXISTS `table\_news`

(

`id\_news` SMALLINT(6) NOT NULL AUTO\_INCREMENT,

`news\_title` VARCHAR(80) NOT NULL,

`news\_content` TEXT NOT NULL,

`news\_author` SMALLINT(6) NOT NULL,

PRIMARY KEY(`id\_news`),

FOREIGN KEY(`news\_author`) REFERENCES `table\_users`(`id\_user`)

);

FOREIGN KEY(`news\_author`) REFENCES `table\_users`(`id\_user`); => On a une clé étrangère news\_author qui fait référence à id\_user de table\_users.

DESCRIBE table\_news;

DESCRIBE table\_users;

### Altérer une table

ALTER TABLE `table\_news`

ADD FOREIGN KEY(`news\_author`) REFERENCES `table\_users`(`id\_user`);

On altère la table table\_news. On ajoute une clé étrangère sur news\_author qui fait référence à id\_user dans table\_users.

### Nommer une contrainte

Ce n'est pas nécessaire de renommer une clé étrangère.

ALTER TABLE `table\_news`

ADD CONSTRAINT fk\_jesaispasquoi FOREIGN KEY(`news\_author`) REFERENCES `table\_users`(`id\_user`);

### Supprimer une contrainte

ALTER TABLE `table\_news`

DROP FOREIGN KEY `fk\_jesaispasquoi`;

Remarque : L'intérêt de nommer une classe étrangère est qu'on peut facilement les supprimer.

## Pour la lecture et la recherche de données (INDEX)

Au niveau des moteur de stockage, le moteur MyISAM est très pratique quand on a une table avec beaucoup de lecture, de recherches à faire.

Quand on a beaucoup de recherches à faire, l'idéal est de définir un index. Sur un système, où il n'y aurait pas cache, SQL devrait parcourir toutes les données et puisque cela fonctionne comme une structure d'arbre (tronc, division en 2 puis en 2, etc), on peut avoir une grande quantité de données qui pourrait ralentir le temps d'exécution des requêtes.

C'est comme un index Windows qui va indexer les fichiers sur notre système d'exploitation. L'avantage est qu'on accélère les choses puisqu'on aura un fichier (un cache) qui va enregistrer toute l'arborescence de notre table. Ensuite quand on a besoin de faire de la recherche ou de la sélection, tout ira beaucoup plus vite lorsqu'on a déjà tout indexé pour donner les résultats de notre requête (la vitesse d'exécution peut être multiplié par 3, même 20 parfois).

C'est mieux généralement de le faire sur MyISAM plutôt que InnoDB. Notre table table\_users est par défaut sur InnoDB mais l'idéal dans ce cas est de l'avoir sur MyISAM.

### Créer un index (INDEX)

Cette requête doit être faite sur une table existante.

CREATE INDEX `idx\_users` on `table\_users`(`user\_name`, `user\_email`);

Cet index va rendre plus rapide les requêtes de sélection, c’est-à-dire de recherche.

*Remarque : DESCRIBE table\_users; et SHOW TABLE STATUS; ne donnent pas d'indication sur les clés étrangères.*

### Supprimer un index

ALTER TABLE `table\_users`

DROP INDEX `idx\_users`;

### Supprimer les tables liées

Il faut casser les relations (clés étrangères) entre les tables, avant de pouvoir supprimer les tables.

On ne peut pas supprimer une table avec des relations existantes.

Remarque : on parle de suppression en cascade lorsqu'un utilisateur souhaite supprimer son identifiant et tous les articles liés.

On retire la table avec la clé étrangère pour commencer.

DROP TABLE table\_users;

DROP TABLE table\_news;

En dehors d'un vrai projet, on peut le faire comme ici en local mais c'est à éviter sur table avec plein de données.

## Résumé

Ce qu'il faut connaitre

NOT NULL colonne obligatoire.

DEFAULT valeur par défaut.

CHECK vérification d'une valeur.

UNIQUE valeur de champ unique.

PRIMARY KEY clé primaire.

AUTO\_INCREMENT incrémentation de 1 en 1.

FOREIGN KEY clé étrangère.

INDEX index pour la lecture et la recherche.

On a vu les syntaxes pour MySQL général. On a des syntaxes que pour MySQL, Oracle, PostgreSQL, SQL server.

## Bonus

Au moment de la création de table, spécifier des informations précises sur le moteur de la table et de l'encodage (autre que ceux par défaut).

Le moteur par défaut est InnoDB mais dans le cas de clés étrangères, il est préférable d'avoir une table en MyISAM.

CREATE TABLE IF NOT EXISTS `table\_users`

(

`id\_user` SMALLINT(6) NOT NULL AUTO\_INCREMENT,

`user\_name` VARCHAR(30) NOT NULL,

`user\_email` VARCHAR(100) NOT NULL,

`user\_registration` DATE NOT NULL,

`user\_admin` BOOLEAN DEFAULT 0,

PRIMARY KEY(`id\_user`)

)

ENGINE = MyISAM

CHARACTER SET utf8mb4

COLLATE utf8mb4\_unicode\_ci;

*Remarque : \_ci : insensible à la case.*

SHOW TABLE STATUS;

La base de données est maintenant à MyISAM alors que par défaut c'est InnoDB. On regarde également si ce sont les bons encodages.

L'intérêt est de préparer une table avec les bonnes informations d'encodage et de moteur pour le jour où on utilise un autre ordinateur avec MySQL paramétré différemment.

Spécifier manuellement ses paramètres est recommandé à chaque fois que l'on fait ses tables de données.

# Les utilisateurs et privilèges

SQL #8 - utilisateurs et privilèges : <https://www.youtube.com/watch?v=IvOCAobeihs>

Dans cette vidéo, vous saurez comment gérer des utilisateurs en SQL et leur spécifier des droits sur les actions qu'ils peuvent faire ou non sur les bases, tables et champs de table.

root est l'utilisateur par défaut quand on se connecte en local (pas de problème lorsqu'on ne travaille pas sur internet). Cet utilisateur possède tous les droits (créer et supprimer des bases, des tables, gérer les droits des autres utilisateurs).

Lorsqu'on travaille via un hébergeur en ligne, on reçoit un utilisateur avec des droits limités, appelé des privilèges.

Privilèges = limite de droit d'utilisation.

Ici on s'intéresse à la gestion des utilisateurs et la gestion de leurs droits.

## Gestion des utilisateurs

### Afficher la liste des utilisateurs par défaut

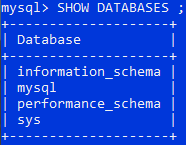
#### Se connecter avec l'utilisateur root

mysqld.exe --console # démarrer le serveur (dans un terminal à part).

mysql.exe -u root –p # démarrer MySQL.

#### Afficher les bases de données

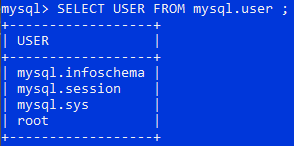
SHOW DATABASES ;



La base de données mysql (ce sera une autre base sur d'autres systèmes) contient une table users qui permet de récupérer toutes les informations des utilisateurs dont l'utilisateur root.

#### Afficher la table des utilisateurs

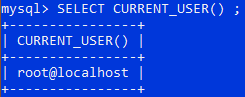
SELECT USER FROM mysql.user ;



Cette commande de recherche affiche la liste des utilisateurs par défaut sur MySQL

### Afficher l'utilisateur en cours <Utilisateur@nomD'hôte>

SELECT CURRENT\_USER() ;



Cette fonction de MySQL permet de retourner l'utilisateur courant, celui qui est utilisé. Le retour donne le nomDeL'utilisateur@nomD'hôte, c'est dire l'hôte sur lequel on va se connecter. Le standard local est local host mais si on est en ligne, on obtient soit une adresse IP précise soit un nom de domaine. Localhost a une adresse IP propre mais on utilise le nom, plus facile à écrire que 127.0.0.1

### Créer un utilisateur

Si l'utilisateur existe déjà, on obtient une erreur ou un avertissement. On crée l'utilisateur sous forme de chaîne ``@`` pour éviter des problèmes liés aux caractères spéciaux. On lui attribue directement un mot de passe avec la commande IDENTIFY BY '' mais on peut aussi l'ajouter par la suite.

CREATE USER IF NOT EXISTS `jacquets`@`localhost` IDENTIFIED BY 'motDePasse' ;

On affiche, on se déconnecte et on se reconnecte au nouvel utilisateur.

SELECT USER FROM mysql.user ;

exit

mysql.exe -u jacquets -p

Confirmer qu'on est bien sur le nouvel utilisateur.

SELECT CURRENT\_USER() ;

### Modifier le mot de passe

#### Les anciennes syntaxes (SET)

Cette synthase n'est plus conseillée.

SET PASSWORD FOR `jacquets`@`localhost` = 'pass' ;

On peut aussi lui imposer un hash pour éviter de la stocker en clair mais la fonction PASSWORD() n'existe plus depuis la version 5.5.

SET PASSWORD FOR `jacquets`@`localhost` = PASSWORD('pass') ;

Cette syntaxe n'est pas recommandée. Aujourd'hui c'est la syntaxe avec l'altération qui est le plus recommandé.

#### La syntaxe privilégiée (ALTER)

La meilleure syntaxe pour le changement de mot de passe.

ALTER USER `jacquets`@`localhost` IDENTIFIED BY 'newPass' ;

Se déconnecter pour tester si le changement a bien fonctionné.

exit

mysql.exe -u root -p

### Modifier le nom d'utilisateur (RENAME)

En étant connecté à root,

Syntaxe qui reprend toutes les informations précédemment, TO puis les nouvelles informations. Elle nécessite d'être connecté avec un autre utilisateur.

RENAME USER `jacquets`@`localhost` TO `js`@`localhost`;

Afficher pour vérifier.

SELECT USER FROM mysql.user ;

### Supprimer un utilisateur

DROP USER `js`@`localhost` ;

Afficher pour vérifier.

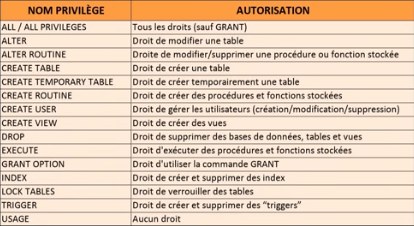
SELECT USER FROM mysql.user ;

## Gestion des droits/privilèges

On utilise très peu et de manière sommaire cette gestion de droits et privilèges.

### Modifier les droits

#### Types de privilèges à un utilisateur – GRANT



GRANT est une commande qui prend certains privilèges. Par exemple, ALL est un nom de privilège qui permet d'avoir tous les droits. root a le privilège ALL. On peut gérer tout particulièrement, si on ne met pas CREATE TABLE, l'utilisateur n'aura pas le droit de créer de table. Si on ne veut pas que plusieurs milliers d'utilisateurs non-administrateur n'ait pas l'autorisation de créer des utilisateurs avec CREATE USER.

ALL c'est tous les droits sauf GRANT c’est-à-dire qu'il ne permet pas d'utiliser la commande pour changer les privilèges. Pour cela, il faut utiliser GRANT OPTION.

USAGE ce n'est pas pour modifier les droits, aucun droit.

L'application de privilège, c'est GRANT suivi des privilèges en particulier. Par exemple, on peut ajouter plusieurs privilèges séparés par une virgule (GRANT CREATE TABLE, etc.), donner le privilège de SELECT (GRANT SELECT etc.).

#### Accorder les droits sur toutes les tables et toutes les bases de données sauf pour les droits d'utilisateurs

Le plus souvent, on fait un GRANT ALL pour faire beaucoup de choses sauf de modifier les droits. On crée ainsi un utilisateur qui ne pourra pas modifier les droits.

On donne le droit de tout faire sauf de donner le droit aux autres, de modifier les droits de root et des autres utilisateurs. Il peut aussi tout faire sur sa base de données, de faire des tables, faire et supprimer des enregistrements, de SELECT, etc. Il gère sa base de données. On donne ici le droit sur toutes les tables et toutes les bases avec le symbole (joker) étoile (\*). La première étoile (\*) représente toutes les bases de données et la secondes toutes les tables.

GRANT ALL ON \*.\* ;

#### Accorder privilèges sur une seule base

Cet utilisateur pourra tout faire sur sa base mais pas sur les autres bases.

GRANT ALL ON jsbase.\* ;

#### Accorder privilèges sur la table d'une base

Cet utilisateur pourra tout faire sur une seule table.

GRANT ALL ON jsbase.jstable ;

Où accorder les permissions juste sur le nom d'une table.

GRANT ALL ON jstable ;

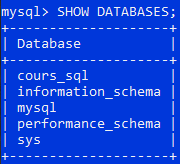
#### Accorder privilèges sur le champ d'une table

Cet utilisateur pourra tout faire sur un seul champ.

GRANT ALL ON jstable.username ;

#### Accorder tous les droits sur cours\_sql.\* à `jacquets`@`localhost`

On choisit d'accorder tous les droits de la base cours\_sql.



Accorder les droits uniquement sur une base de données.

GRANT ALL

ON cours\_sql.\*

TO `js`@`localhost` ;

… avec des droits limités. il ne pourra pas créer/supprimer de nouvelles bases, accéder à de nouvelles bases de données.

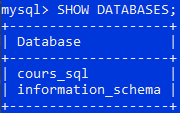
Ensuite se déconnecter de root pour tester si le changement a bien fonctionné.

exit

mysql.exe -u js -p

On ne peut pas voir les autres bases et encore moins accéder à d'autres bases.

SHOW DATABASES;



USE mysql;



#### Accorder tous les privilèges et le droit de changer les privilèges

Accorder les droits uniquement sur sa base de données.

GRANT ALL

ON cours\_sql.\*

TO `js`@`localhost`

WITH GRANT OPTION ;

… avec le droit d'utiliser les commandes GRANT pour changer les privilèges (la commande GRANT).

#### Accorder tous les droits mais retirer le droit de créer des tables

En étant connecté à root,

GRANT ALL

ON cours\_sql.\*

TO `js`@`localhost`;

REVOKE CREATE

ON cours\_sql.\*

FROM `js`@`localhost`;

On peut accorder tous les droits mais retirer certains privilèges 1 à 1 si l'on veut.

#### Retirer tous les droits

REVOKE ALL

ON cours\_sql.\*

FROM `js`@`localhost` ;

On retire les droits sur cours\_sql de (FROM) l'utilisateur `js`@`localhost`.

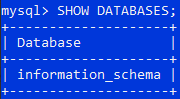
Ensuite se déconnecter de root pour tester si le changement a bien fonctionné.

exit

mysql.exe -u js -p

On ne peut pas voir les autres bases et encore moins accéder à d'autres bases.

SHOW DATABASES;



… on ne peut plus créer de base non plus.

CREATE DATABASE test;



… on ne peut pas créer d'utilisateur.

CREATE USER IF NOT EXISTS `toto`@`localhost` IDENTIFIED BY '' ;



En effet, il n'a pas le privilège CREATE USER.

# Lecture de données (data1.sql)

SQL #9 - lecture données :

https://www.youtube.com/watch?v=QfN\_URkzw94&list=PLrSOXFDHBtfGl66sXijiN8SU9YJaM\_EQg&index=9

mysqld.exe --console # démarrer le serveur (dans un terminal à part).

mysql.exe -w root –P # démarrer MySQL.

SOURCE c:/users/Jason/Desktop/FV/sql\_data1.sql ; # chemin vers notre fichier source. MySQL utilise des slashs. Ne pas hésiter à exécuter la commande plusieurs fois.

SHOW TABLES ; # vérifier les données importées.

DESCRIBE fv\_products ; # vérifier les données importées.

DESCRIBE fv\_users ; # vérifier les données importées.

SELECT product\_name , product\_amount FROM fv\_products ; # SELECT pour récupérer les données d'une base.

SELECT \* FROM fv\_products ; # récupérer toutes les infos avec \*.

# Note : La vitesse de récupération dépend de notre base de données enregistrée au niveau de notre moteur de stockage : Si on a plein de lecture, on utilisera MyISAM et si on a plein de update et des delete, on utilisera InnoDB.

SELECT \* FROM fv\_users ; # récupérer toutes les infos de notre table.

SELECT DISTINCT product\_name FROM fv\_products ; # Lire les données en écartant les doublons. DISTINCT ou UNIQUE pour la base de données Oracle.

SELECT DISTINCT product\_name AS Pn FROM fv\_products ; # Création d'un alias Pn de mon champ product\_name.

SELECT DISTINCT product\_name AS Nom\_du\_produit FROM fv\_products ; # Création d'un alias (plus explicite).

SELECT DISTINCT product\_name AS "Nom du produit" FROM fv\_products ; # Création d'un alias (sans underscore).

SELECT DISTINCT product\_name AS "Nom du produit" FROM fv\_products AS Inventory ; # Création de deux alias.

exit

./mysqladmin.exe -u root -p shutdown

exit

# Ensuite voir la fenêtre du serveur.

exit

# Filtrer les requêtes (data2.sql)

# Serveur MySQL démarré.

# Ouvrir un autre terminal pour le client :

./mysql.exe -u root -p

SOURCE c:/users/Jason/Desktop/FV/data.sql ; # chemin vers notre fichier source.

SOURCE c:/users/Jason/Desktop/FV/data.sql ; # exécuter plusieurs fois.

SOURCE c:/users/Jason/Desktop/FV/data.sql ; # exécuter plusieurs fois.

SHOW DATABASES ; # Voir notre base "fv\_databases".

use fv\_databases ;

SHOW TABLES ; # montrer la table.

select \* from fv\_games ; # lire la table.

select \* from fv\_games WHERE id\_game = 11 ; # faire des filtres avec la clause where et un opérateur de comparaison (=, !=, <, >, <=, >=).

select \* from fv\_games WHERE id\_game = 11 OR id\_game = 6 ; # faire des filtres avec des opérateurs pour cumuler des tests (AND, OR).

select \* from fv\_games WHERE game\_price > 5 AND game\_price < 30 ;

select \* from fv\_games WHERE game\_price BETWEEN 5 AND 30 ;

select \* from fv\_games WHERE game\_stock = 2 OR game\_stock = 5 OR game\_stock = 6 ;

select \* from fv\_games WHERE game\_stock IN(2, 5, 6) ; # mettre des guillemets ou double guillemets pour les chaînes.

select \* from fv\_games WHERE game\_tittle LIKE 'Pokemon%' ; # le pourcent remplace un ou plusieurs caractères qui vient après.

select \* from fv\_games WHERE game\_tittle LIKE 'Tekken\_' ; # l'underscore remplace un caractère unique qui suit ma chaîne (un nombre ou une lettre).

select \* from fv\_games WHERE game\_price IS NOT NULL ;

select \* from fv\_games WHERE game\_price IS NULL ;

select \* from fv\_games ORDER BY id\_game ; # Par défaut.

select \* from fv\_games ORDER BY game\_tittle ; # Tri par ordre alphabétique.

select \* from fv\_games ORDER BY game\_tittle ASC ; # Tri par ordre ascendant.

select \* from fv\_games ORDER BY game\_tittle DESC ; # Tri par ordre descendant.

select \* from fv\_games ORDER BY game\_tittle, game\_price DESC ; # 'Si des jeux portent le même nom, il seront triés par ordre de prix décroissant.

select \* from fv\_games LIMIT 5 ; # Ne limite pas le nombre d'enregistrement à récupérer mais limite le nombre d'affichage (p.e. 5000 news sur un site web).

select \* from fv\_games LIMIT 5 ; #

select \* from fv\_games LIMIT 5 OFFSET 3 ; # Offset permet de commencer à offset+1 c'est-à-dire depuis le premier qui est zéro.

select \* from fv\_games LIMIT 3, 5 ; # Offset en premier et ensuite le nombre d'enregistrement à retourner.

# Syntaxe différentes suivant le SGBD utilisé.

# LIMIT (MySQL et PostgreSQL) ou TOP (SQLserver, MS Access) ou ROWNUM (Oracle).

# Enregistrer données

./mysql.exe -u root -p

SOURCE c:/users/Jason/Desktop/FV/data.sql ;

SOURCE c:/users/Jason/Desktop/FV/data.sql ;

SOURCE c:/users/Jason/Desktop/FV/data.sql ;

use fv\_databases ;

show DATABASES ;

describe fv\_clients ;

select \* from fv\_clients ; # aucun client enregistré car la table est vide.

use fv\_clients ;

INSERT INTO fv\_clients(client\_firstname, client\_lastname, client\_birthday, client\_loyalty) ;

VALUES('Samuel', 'Jacquet', '1987-11-21', '17') ;

select \* from fv\_clients ; # un client enregistré.

INSERT INTO fv\_clients(client\_firstname, client\_lastname, client\_birthday, client\_loyalty) ;

VALUES

('Samuel', 'Jacquet', '1987-11-21', '17'),

('Samuel', 'Jacquet', '1987-11-21', '17'),

('Samuel', 'Jacquet', '1987-11-21', '17'),

('Samuel', 'Jacquet', '1987-11-21', '17'),

('Samuel', 'Jacquet', '1987-11-21', '17') ; # cinq clients enregistrés.

# Modifier et supprimer

### 4 opérations élémentaires que l'on peut faire sur les données :

## CRUD :

# create (INSERT)

# read (SELECT)

# update (UPDATE)

# delete (DELETE) : modifier les informations.

./mysql.exe -u root -p

Use fv\_database ;

show database ;

show tables ;

select \* from fv\_games ;

update fv\_games set game\_stock = 6, game\_price = 11.59 where id = 1 ; # éditer 1 enregistrement (clause Where).

update fv\_games set game\_stock = 6, game\_price = 11.59 ; # Sans la clause where, on modifie tous les enregistrements.

delete from fv\_games where game\_tittle = 'Tekken' ; # Supprimer les jeux Tekken de la table.

delete from fv\_games ; # Supprimer toute la table.

truncate table fv\_games ; # requête prévue pour vider une table.

# Les fonctions d'agrégation

# sur base du fichier data2 (Les jeux vidéo).

select count(\*) from fv\_games ; # Retourner le nombre d'enregistrement.

select count(\*) as total\_game from fv\_games ; # Retourner un champ nommé.

select avg(game\_price) from fv\_games ; # Retourner la moyenne des prix.

select max(Game\_price) from fv\_games ; # Retourner le jeux le plus cher (utiliser sur les valeurs numériques mais aussi les chaînes de caractères.)

select min(game\_price) from fv\_games ;

select sum(game\_stock) from fv\_games ;

# Grouper les résultats

SQL #14 - Grouper les résultats

Connection

mysqld.exe –console # connection au serveur.

mysql.exe -u root -p # connexion avec le client.

Importer le fichier

source C:/Users/Jason/Desktop/FV/sql/data4.sql ; # lancer la requête plusieurs fois pour être sur que ca travaille.

describe fv\_users ; # retourner toutes les informations qui sont dans le fichier.

select \* from fv\_users ; # retourner tous les enregistrements du fichier.

Retourner le nombre de personnes parmi les utilisateurs avec la commande WHERE

select count(\*) from fv\_users where user\_country = 'France' # Erreur : la clause where ne récupère que les résultats sans plus. -> utiliser la commande GROUPE BY.

Regrouper les résidents par rapport a chacun des pays avec la commande GROUP BY

select count(\*), user\_country

from fv\_users

group by user\_country ; # grouper par rapport a tous les pays de ma table.

Note : La commande GROUP BY est souvent utilisée avec des fonctions d'agrégations (COUNT, MAX, MIN, SUM, AVG) pour grouper les résultats.

Idem en renommant la colonne

select count(\*) as 'Nombre de resident', user\_country

from fv\_users

group by user\_country ; # grouper les résultats de colonnes distincte grâce a un paramètre bien précis.

Regrouper la somme de tous les ages dans le regroupement par pays et la somme totale des tous les ages avec la commande WITH ROLLUP

select user\_country, sum(user\_age)

from fv\_users

group by user\_country with rollup ; # retourne la somme de tous les âges dans un regroupement par pays et a la fin, la somme total de tous les ages.

Filtrer les résultats d'un groupement de donnes avec la commande HAVING

select user\_country, sum(user\_age)

from fv\_users

group by user\_country

having sum(user\_age) > 20 ; # retourne toute la somme des âges qui dépassent 20 (filtrer le retour de notre requete).

# unions et intersections

SQL #15 – unions et intersections

mysql.exe -u rooter-p # se connecter au serveur.

source C:/Users/Jason/Desktop/FV/sql/data5.sql ; # lancer la requete plusieurs fois pour etre sur que ca travaille.

use fv\_database ;

select \* from fv\_site1\_users;

select \* from fv\_site2\_users; # recuperer les noms d'utilisateurs des 2 tables.

# union = permet de concaténer le résultat de 2 requêtes.

select \* from fv\_site1\_users

union

select \* from fv\_site2\_users; # les doublons (enregistrement purement identique sur toutes les colonnes) seront écartés.

# règles :

#1# le même nombre de colonne à récupérer.

#2# récupérer les colonnes dans le même ordre.

#3# récupérer les même type de données (int, char, date).

select user\_name from fv\_site1\_users

union

select user\_name from fv\_site2\_users; # les doublons (uniquement pour la colonne utilisateur) seront écartés.

select user\_name from fv\_site1\_users

union all

select user\_name from fv\_site2\_users; # les doublons seront pris en compte.

select user\_name from fv\_site1\_users

intersect

select user\_name from fv\_site2\_users; # la requête retourne que les résultats affichés en double.

select user\_name from fv\_site1\_users

minus # 'exept' sur d'autre fgbd que Oracle.

select user\_name from fv\_site2\_users; # la requête retourne les résultats de la première tables non présent dans la seconde table.

#Minus, c'est les utilisateurs de la première table moins ceux présent dans la seconde table.

select \* from fv\_site1\_users

union

select \* from fv\_site2\_users order by user\_name; # classé par ordre alphabétique. Retourne également les doublons tant que les champs ne sont pas tous différents.

select user\_name, user\_country from fv\_site1\_users

union

select user\_name, user\_country from fv\_site2\_users order by user\_name; # ici tous les doublons sont écartés classés par ordre alphabétique (appliqué sur toute la requête). La clause d'ordre est appliquée sur toute la concaténation.

# sous-requête

SQL #16 - sous-requête

Se servir du résultat d'une requête dans la clause d'une seconde requête.

mysql.exe -u root -p

source C:/Users/Jason/Desktop/FV/sql/data6.sql;

use fv\_database;

describe fv\_players;

describe fv\_playertrades;

select \* from fv\_players;

select \* from fv\_ playertrades;

1. **Via un opérateur (<, >, =) on peut valider notre sous requête par une clause WHERE.**

# On cherche un vendeur dans la table ci-dessous dont le nom est Icekissy et qui a déjà vendu un jeu ? Si un résultat s'affiche alors oui.

# On se limite à 1 colonne dans le retour des résultats.

# La sous-requête est testée avant de valider une clause (WHERE ou HAVING) suivi d'un opérateur (=, <, >) et enfin on sélectionne des informations (fonctionnement à l'envers).

select \* from fv\_players

where player\_name =

(

select playertrade\_seller

from fv\_playertrades

where playertrade\_seller = 'Icekissy'

limit 1

) ;

# Par contre, l'utilisateur Nephion n'as jamais rien vendu.

select \* from fv\_players

where player\_name =

(

select playertrade\_seller

from fv\_playertrades

where playertrade\_seller = 'Nephion'

limit 1 # limité à un car on ne peux pas comparer plusieurs résultats (sous-requête) avec une seule valeur (requête).

) ;

1. **L'opérateur IN permet de récupérer plusieurs enregistrements de la requête imbriquée.**

# Utiliser IN a la place de l'opérateur afin de retourner quand même un résultat si la sous-requête retourne plus d'une ligne.

SELECT \* FROM fv\_players

WHERE player\_name IN

(

SELECT playertrade\_seller

FROM fv\_playertrades

WHERE playertrade\_seller = 'Nephion'

LIMIT 1 # limité à un car on ne peut pas comparer plusieurs résultats (sous-requête) avec une seule valeur (requête).

) ;

# Retourne la liste des vendeurs, les informations des joueurs qui ont vendus des objets.

# On récupère plusieurs informations comparées une par une avec l'opérateur IN.

SELECT \* FROM fv\_players

WHERE player\_name IN

(

SELECT playertrade\_seller

FROM fv\_playertrades

) ;

1. **ANY ou SOME vaux au moins pour une égalité de la sous requête**

# Valider le premier select si l'égalité à la sous requête correspond à au moins un enregistrement de mon second select.

# = ANY signifie que l'on vérifie qu'il y a au moins une égalité de cette sous requête. Vérifie si il y a au moins un cas.

SELECT \* FROM fv\_players

WHERE player\_name = ANY

(

SELECT playertrade\_seller

FROM fv\_playertrades

) ;

# retourne un résultat s'Il existe des vendeurs qui ont vendus des épées en mousse à quelqu'un. La clause WHERE fait que ça doit correspondre à au moins une valeur de notre sous-requête.

SELECT \* FROM fv\_players

WHERE player\_name = ANY

(

SELECT playertrade\_seller

FROM fv\_playertrades

WHERE playertrade\_item = 'Epée en mousse'

) ;

# SOME est un alias de ANY.

SELECT \* FROM fv\_players

WHERE player\_name = SOME

(

SELECT playertrade\_seller

FROM fv\_playertrades

WHERE playertrade\_item = 'Epée en mousse'

) ;

1. **EXISTS permet de réaliser une requête sous la condition que la sous requête a retourné quelque chose.**

# Vérifie si la sous requête renvoie à un résultat avant de traiter la requête sans autre filtre.

SELECT \* FROM fv\_players

WHERE EXISTS # Si la sous-requête a des résultats qui existent, la clause WHERE renvoie True et dans ce cas on sélectionne tous les joueurs.

(

SELECT \* # selection de tous les vendeurs.

FROM fv\_playertrades # récupérer toutes les infos de la table des échanges.

WHERE playertrade\_item = 'Epée en mousse' # récupérer dans le cas où l'objet de la transaction est une épée en mousse.

) ;

# retourne tous les vendeurs de la table des échanges dont l'objet des transactions est une épée en mousse.

SELECT \*

FROM fv\_playertrades

WHERE playertrade\_item = 'Epée en mousse'

# Vérifie si la sous requête renvoie à un résultat avant de traiter la requête avec un filtre (>10) et en ordre croissant.

SELECT \* FROM fv\_players

WHERE EXISTS

(

SELECT \*

FROM fv\_playertrades

WHERE playertrade\_item = 'Epée en mousse'

)  AND player\_level > 10

ORDER BY player\_level ASC;

# Vérifie si la sous requête renvoie à un résultat avant de traiter la requête avec un filtre (>10) et en ordre croissant.

SELECT \* FROM fv\_players

WHERE EXISTS

(

SELECT \*

FROM fv\_playertrades

WHERE playertrade\_item = 'Epée en bois'

)  AND player\_level > 10

ORDER BY player\_level ASC;

# jointures

SQL #17 - jointures

Rappel sur les clés étrangères : c'est créer un lien entre 2 tables qui ont une colonne d'identifiant en commun. Le champ d'une table A est reliée au champ d'une table B. Il faut utiliser le moteur de stockage InnoDB avec la table qui utilise ces cl2s étrangère. Ça ne marchera pas avec MyISA. Contrairement au cours précèdent, on crée une relation avec une clé étrangères.

Rendre cohérent la structure de données en stockant des entiers, des clés, des identifiants pour créer les différentes relations entre plusieurs tables. Dans ce fichier data7 (new), on cherche a relier 2 tables : échanges entre une table de joueurs et une table d'échanges (transactions entre joueurs). On pourrait encore créer une table d'objet qui serait liee a notre table de transaction d'objets. On pourrait également faire une table qui contient la liste des classes, ce qui évite d'avoir de la redondance entre les classes mais ouvre également le spectre des requêtes possibles pour récupérer ces informations tel que faire la liste des classes dans le jeu plutôt que de passer par une table avec des doublons.

Le principe des identifiants est d'attribuer une valeur d'identifiant INT (une clé étrangère remplace des chaines de caractères) au champ de ma table. Par exemple `player\_seller` et `player\_buyer` seront des identifiants de joueurs `id\_player`. Il y a autant de tables que d'identifiants de joueurs auquel la table se rapporte.

CREATE TABLE IF NOT EXISTS `fv\_players`

(

`id\_player` INT NOT NULL AUTO\_INCREMENT,

`player\_name` VARCHAR(65) NOT NULL UNIQUE,

`player\_level` INT NOT NULL,

`player\_ability` INT NOT NULL,

PRIMARY KEY(`id\_player`)

) ;

mysql.exe -u root -p

source C:/Users/Jason/Desktop/FV/sql/data7.sql;

use fv\_database;

show tables;

select \* from fv\_players;

select \* from fv\_items;

describe fv\_abilities;

Note : organisation des informations entre les différentes tables, on peut ajouter à tout moment de nouvelles classes, de nouveaux joueurs peuvent s'inscrire, de nouveaux objets au jeu, de nouveau échanges entre joueur grâce à des agencement avec des tables spécifiques.

Pratique pour l'évolutivité (la maintenance) de la base de données.

SELECT \* FROM fv\_players; # récupérer la liste des joueurs.

SELECT \* FROM fv\_abilities; # récupérer les classes des joueurs.

## 1 ## Premier type de jointure : Les informations des deux tables doivent être valides grâce à une jointure interne.

# Retourner tous les joueurs et les classes qui correspondent.

SELECT player\_name, player\_level, ability\_name

FROM fv\_players # Premiere table au quel on cree une jointure pour relier les 2 tables (table des playeurs a la table des habilitées).

INNER JOIN fv\_abilities ON fv\_players.player\_ability = fv\_abilities.id\_ability; # Seconde table au quel on fait correspondent la relation entre une cle etrangere et une cle primaire (tel champ d'une table = tel champ d'une autre table).

# idem avec la clause WHERE pour indiquer la relation (attention la jointure est plus explicite et permet ensuite de filtrer avec une clause WHERE)

SELECT player\_name, player\_level, ability\_name

FROM fv\_players, fv\_abilities

WHERE fv\_players.player\_ability = fv\_abilities.id\_ability;

# Retourner tous les joueurs et les classes qui correspondent avec une clause WHERE et un classement ORDER BY.

SELECT player\_name, player\_level, ability\_name

FROM fv\_players

INNER JOIN fv\_abilities ON fv\_players.player\_ability = fv\_abilities.id\_ability;

WHERE player\_level > 10;

ORDER BY player\_level ASC;

## 2 ## Second type de jointure : jointure naturelle dans le cas ou la relation se fait entre deux cles qui portent le meme nom de colonne et le meme type de donnees.

# retourner une jointure naturelle entre deux tables.

SELECT \*

FROM fv\_players

NATURAL JOIN fv\_abilities ON fv\_players.id = fv\_abilities.id ;

## 3 ## Troisieme type de jointure (non prit en charge par MySQL) : jointure gauche et externe LEFT JOIN ou LEFT OUTER

# Retourner tous les enregistrements de la premiere table (from) meme si il n'y a pas d'information qui existe dans la seconde (left join).

# Retourner la liste de toutes les classes de mon jeu meme si il y a des joueurs qui ne se servent pas de l'une d'entre elles.

SELECT \*

FROM fv\_abilities

LEFT JOIN fv\_players ON fv\_abilities.id\_ability = fv\_players.player\_ability;

# Ne retourne pas les enregistrements qui n'existent pas dans les deux cas.

SELECT \*

FROM fv\_abilities

INNER JOIN fv\_players ON fv\_abilities.id\_ability = fv\_players.player\_ability;

# Inverse de INNER JOIN

# Retourne ce qui n'as pas d'enregistrement dans les deux tables.

# Filtrer le résultat dans les cas, parmi la sélection (select) où il n'y a pas de joueurs attribues à ces classes.

SELECT \*

FROM fv\_abilities

LEFT JOIN fv\_players ON fv\_abilities.id\_ability = fv\_players.player\_ability

WHERE fv\_players.player\_ability IS NULL;

# Retourner les classes qui ne sont pas utilisées pas les joueurs.

SELECT ability\_name

FROM fv\_abilities

LEFT JOIN fv\_players ON fv\_abilities.id\_ability = fv\_players.player\_ability

WHERE fv\_players.player\_ability IS NULL;

# Retourner la liste des joueurs sans classes (situation inverse).

SELECT ability\_name

FROM fv\_abilities

RIGHT JOIN fv\_players ON fv\_abilities.id\_ability = fv\_players.player\_ability

WHERE fv\_players.player\_ability IS NULL;

# Retourner la liste des objets même ceux qui n'ont pas été utilisés dans des échanges.

SELECT item\_name

FROM fv\_items

LEFT JOIN fv\_playertrades ON fv\_items.id\_item = fv\_playertrades.playertrade\_item;

# Retourner la liste des objets qui n'ont pas été utilisés dans des échanges.

SELECT item\_name

FROM fv\_items

INNER JOIN fv\_playertrades ON fv\_items.id\_item = fv\_playertrades.playertrade\_item;

# Retourner la liste (tester)….

SELECT item\_name

FROM fv\_items

LEFT JOIN fv\_playertrades ON fv\_items.id\_item = fv\_playertrades.playertrade\_item

WHERE fv\_playertrades.playertrade\_item IS NULL;

# Retourner la totalité des informations (JOINTURE EXTERNE PLEINE). FULL JOIN n'est pas possible sur MySQL.

SELECT item\_name

FROM fv\_items

FULL JOIN fv\_playertrades ON fv\_items.id\_item = fv\_playertrades.playertrade\_item;

# Retourner la totalité des informations sur MySQL.

SELECT \*

FROM tableA

UNION SELECT \* FROM tableB

# Retourne le produit cartésien des deux tables.

# Chaque champ de ma tableA va être relie à chaque champ de ma tableB.

# Multiplie les enregistrements d'une table par l'autre.

SELECT \*

FROM fv\_players

CROSS JOIN fv\_abilities;

# idem et trier.

SELECT \*

FROM fv\_players

CROSS JOIN fv\_abilities

ORDER BY player\_name;

# Jointure interne : il faut que l'information existe dans les deux tables.

# Jointure naturelle : il faut qu'il y ai au moins une colonne de même nom et de même type (non recommandée).

# Jointure externe gauche : il retourne toutes les classes mais qui ne sont pas utilisées par les joueurs (On n'inclue pas B).

# fonctions chaines de caractères

SQL #18 – fonctions chaines de caractères

Utilisation de MySQL :

* Interface de connexion a des données depuis un langage de programmation : exemple de l'environnement Web avec PHP pour interroger une base.
* Interface de connexion avec des applications mobiles (Utilisation de SQLite).
* Pratiquement jamais utilise seul sauf chez les gestionnaires de bases de données qui n'ont jamais vu de langage de programmation.

Exemple de SGBD(R) : MySQL, Oracle DATABASE PostgreSQL, SQLite, MS SQL Server.

mysql.exe -o root -p

SOURCE C:/Users/jachampagne/Desktop/FV/sql/data8.sql;

use fv\_database;

SELECT \* FROM fv\_carbrands;

SELECT \* FROM fv\_countries;

#1# Récupérer le nombre de caractères d'une chaine.

SELECT carbrands\_name, CHAR\_LENGTH(carbrand\_name) # character\_length est une fonction alias sur MySQL. LEN pour POST SERVER de miscrosoft et LENGTH pour d'autres type de SGBD.

FROM fv\_carbrands;

#2# Concatenation de chaines ('string'&' '&'string2')

SELECT CONCAT(carbrands\_name, ' (',carbrand\_country, ')') # SELECT CARBRANDS || ' (', carbrand\_country, ')') sur POSTGRE SQL par exemple.

FROM fv\_carbrands;

#3# Conversion en minuscule ou majuscule avec lower() ou upper().

SELECT LOWER(carbrans\_name) FROM fv\_carbrands;

SELECT UPPER(carbrans\_name) FROM fv\_carbrands;

#4# Inversion de chaine.

SELECT REVERSE(carbrans\_name) FROM fv\_carbrands;

#5# Suppression d'espace blancs

SELECT TRIM(' Bonjour '); # Enlever la partie de droite (r-trim) ou de gauche (l-trim) ou d'autres alias.

#6# Remplacer d'une chaine

Utilisation : changement de liens d'url (éviter les erreur 404, liens non valides), modification du nom, de l'extension de notre site ou d'un sous domaine, modification de http en https (protocole sécurisé).

# changer l'element GROUP en GROUPE

UPDATE fv\_carbrands

SET carbrand\_website = REPLACE(carbrand\_website,'group','groupe') # REPLACE(1er élément sur lequel on va travailler, 2eme élément qui doit être remplace, 3eme élément que l'on veux obtenir).

WHERE carbrand\_name = 'Renault';

#7# position d'une occurrence (sous-entendu une chaine) dans une chaine.

SELECT carbrand\_name, LOCATE('yo', carbrand\_name) FROM fv\_carbrands ; # INSTR prend les paramètres inverses de LOCATE(l'information que l'on cherche en premier, la chaine sur lequel on travaille). Retourne 0 s'il ne trouve pas d'occurrence.

#8# extraction d'une sous-chaine

SELECT carbrand\_name, substr(carbrand\_name, 2) FROM fv\_carbrands ; # permet de récupérer une chaine a partir du deuxième caractères.

SELECT substr('Bonjour', 2);

SELECT carbrand\_name, substring(carbrand\_name, 2, 2) FROM fv\_carbrands ; # récupérer 2 caractères après le deuxième caractère.

#9# récupération de la longueur d'une chaine en bit/octet.

Exemple de fonction non vu au cours.

# fonctions dates et heures

SQL #19 – fonctions dates et heures

Chaque fonction est propre à un système de gestion de base de données.

Certains RGBD utilisent des fonctions similaire et d'autre non.

Jeter un œil sur les fonctions non abordées en vidéos propre ou non aux systèmes : MySQL (Oracle), SQLite, PostgreSQL, SQLserver.

Quelques exemples de fonctions sur le RGBD MySQL:

#1# Se connecter au client MySQL.

# Format:

# 10, 2 : représentation d'entiers par des nombres.

# 'Bonjour', "Bonjour" : représentation d'une chaine (formatage de date en chaine).

Date -> string

string -> date

# %d -> extraire le jour d'une date en formatant une information. Sur un autre RGBD, ce ne sont pas forcement les memes infos (exemple : 03).

# %m -> extraire le mois (exemple : 05).

# %Y -> extraire l'année représente par 4 chiffres digitaux (exemple : 2008).

# %h -> représentation de l'heure. Heure définie sur 24h (0 a 23).

# %i -> représentation des minutes. Le m est déjà représenté pour les mois.

# %S -> représentation des secondes.

# Note: pour certains RGBD, la date est représentée au format textuelle (dd, yyyy, etc.). Quels sont les formats d'identification a connaitre pour le formatage des heures et des dates.

#2# Connaitre la date actuelle.

# Application : dater une publication avec MySQL. permet d'enregistrer dans une base de données en mettant la date.

SELECT NOW(); # 2020-03-05 16:30:16

SYSDATE; # est la version d'Oracle pour récupérer le timestamp actuel.

DATE('now'); # est la version PostgreSQL pour récupérer le timestamp actuel. Une fonction DATE() avec un identificateur sous forme de chaine de caractères (mot now).

GETDATE(); est la version SQLserver pour récupérer le timestamp actuel.

#3# Exemple d'utilisation de la fonction now().

INSERT INTO ''

VALUES

('Article', 'Auteur', now()); # Passer la fonction now() pour récupérer la date de publication de l'article.

#4# Extraction d'information (obtenir le jour, le mois, etc.) # pour MySQL ou SQLserver

SELECT DAY('2020-02-13'); # date en dur OU colonne dans une table en base de données.

SELECT MONTH('2020-02-13');

SELECT YEAR('2020-02-13');

SELECT HOUR('2020-02-13 16:45:21');

# pour Oracle et POSTgreSQL

EXTRACT(MONTH FROM TIMESTAMP '2020-02-13'); # date en dur OU colonne dans une table en base de données.

#5# Calculer une différence de dates (connaitre l'ancienneté d'un article, connaitre depuis quand un utilisateur ne s'est plus connecte).

SELECT DATEDIFF('2020-02-04', '2020-02-13'); # calcul de différence en jours.

SELECT AGE('2020-02-04', '2020-02-13'); # pour POSTgreSQL.

SELECT TIMESTAMPDIFF(DAY, '2020-02-04', '2020-02-13'); # permet de choisir l'information a extraire. Pour Oracle.

SELECT DATEDIFF(DAY, '2020-02-04', '2020-02-13'); # pour SQLserveur.

#6# Formatage de date.

#-1- MySQL

DATE\_FORMAT('2020-02-04 14:07:28', '%d/%m/%Y, a %H:%i:%S'); # format de données pour MySQL. 1er paramètre équivaux a une date littérale OU une colonne dans une table de base de données.

#-2- Oracle & PostgreSQL

TO\_CHAR(<DATE>, <FORMAT>); -> format tel que DD/MM/YYYY HH24:MI:SS.

#-4- SQLite

STRFTTIME(<FORMAT>, <DATE>);

#-5- SQL Server

FORMAT(<date>, <format>); -> dd/MM/yyyy.

<https://youtu.be/9Y20_0MQzEE>

# Fonctions Mathématiques

SQL #20 - Fonctions Mathématiques

SQLite -> RGBD avec peu de fonction disponible. Il faut donc en faire sois-mêmes ou passer par un langage de programmation.

MySQL, PostgreSQL, SQL server ont beaucoup de fonctions disponibles.

#1# Calcul de valeur absolue.

#-1- Oracle.

SELECT ABS(-1) FROM DUAL; # DUAL est une table vide car on ne peut pas faire de fonctions sans table.

#-2- MySQL.

SELECT ABS(-1);

#2# Fonctions trigonométriques.

SELECT COS(PI()); # Calculer le cosinus de Pi.

SELECT SIN();

SELECT TAN();

SELECT ACOS();

SELECT ASIN();

SELECT ATAN(); # Arc tangente.

SELECT PI() + 0.0000000; # Spécifier les décimales.

#2# convertir les degrés en radian.

SELECT RADIANS(60); # angle de 60\*C en radians.

SELECT DEGREES(60); # radians 60 en dégrées.

#3# retourner un arrondis

SELECT CEIL(14.26334); # arrondis a l'entier au-dessus.

SELECT FLOW(14.26334); # arrondis a l'entier inferieur.

SELECT ROUND(14.26334); # arrondis a l'entier.

SELECT ROUND(14.26334, 3); #tronque et arrondis a 3 décimales.

SELECT TRUNCATE(14.26334, 2); # tronquer sans faire un arrondi avec un nombre de décimales.

#4# Fonction de puissance.

SELECT POW(2, 3); # fonction de puissance pour la MySQL.

SELECT POWER(2, 3); # fonction de puissance. Alias de MySQL. Dispo sur les autres RGBD.

Base 10 : 0 a 9.

Base 2 : 0 a 1.

Base Hexadecimale : 0 a F.

#5# convertir une base.

SELECT CONV(145, 10, 2); # convertir de la base 10 a la base 2.

#6# Exponentiel et logarythme.

SELECT EX(2); # Fonction exponentielle.

SELECT LN(2); # Fonction de Logarythme Neperien.

SELECT LOG(2, 2); # Logarythme Neperien pour MySQL. Attention, pour les autres RGBD, LOG est le Logarythme en Base.

SELECT LOG10(2); # Alias du Logarythme Neperien pour MySQL mais uniquement en base 10.

#7# Racine carre.

SELECT SQRT(4); # racine carre pour MySQL.

SELECT CBRT(9); # racine cubique pour PostgreSQL.

#8# Calcul de modulo.

# Operateurs : + - \*/ et %.

# 5 % 2 = 1 = reste de la division euclidienne.

#-1- MySQL.

SELECT MOD(5, 2);

SELECT 5 % 2;

SELECT 5 MOD 2;

#9# Valeur en signe.

SELECT SIGN(-3); # retourne -1.

SELECT SIGN(3); # retourne 1.

SELECT SIGN(0); # retourne 0.

#10# Virgule flottante.

SELECT RAND(); # valeur aléatoire a virgule flottante. Valeur comprise entre 0.0 et 1.0.

SELECT RAND(15); # retourne la même valeur avec la même graine, 15 (nombre pseudo aléatoire). Peut être utilise dans le cadre d'une requête pour sélectionner des enregistrements afin de créer un ordre d'affichage aléatoire. Utiliser la fonction rand() après un order by.

SELECT RANDOM(); # nombre entier aléatoire pour SQL server. Le comportement produit par la fonction n'est pas le même que pour MySQL.

# fonctions de sécurité

SQL #21 - fonctions de sécurité

Chiffrement = information (données en clair comme un texte) sur lequel on utilise une clé de chiffrement et de déchiffrement (choisie par un utilisateur) pour obtenir des données chiffrées (On ne parle pas de cryptage). Application : Veracrypt. Peu utilisées avec des bases de données Web. Le chiffrement fait appel à des algorithmes, processus qui dit comment on va utiliser le chiffrement.

Hachage = information (données en clair) qui devient un Hash a la suite d'un processus de chiffrement, c'est a dire après l'appel d'algorithmes. Un Hash est une donnée qui a suivi un processus de hashage, production d'un condense numérique (digital digest). Après avoir produit un hash, il n'est pas possible d'obtenir une donnée en clair. Exemple de PHP, on a un hash et on importe une donnée en clair qui est transformée en Hash. On peut finalement comparer les deux hashs pour voir s'ils sont identiques pendant une identification.

#1# Fonctions de Hashage.

# Utilisée pour la gestion de mdp et l'identification.

# Suivant les RGBDs, il faut passer par des extensions : Transact-SQL pour SQL server ; P/L SQL pour Oracle ; pgcrypto pour PostgreSQL.

#-1- Fonction non sécurisée sur MySQL et PostgreSQL.

# MySQL.

SELECT MD5('Bonjour'); # Fonction qui produit un Hash de 32 caractères (un Hash sur 128 bits) a partir d'une information en clair. En effet, un Hash génère sur N bits donne une chaine de N/4 caractères. Si on stocke un Hash dans un champ d'une colonne d'une de caractères, l'utilisateur choisi un type de données approprie char(32) (taille fixe) mais pas varchar(32) (varchar() sous-entend qu'il peut contenir mois de 32 caractères). La réexécution de SELECT MD5 génère toujours le même Hash de valeur non-aléatoire, la même empreinte numérique d'un mot. Sur d'autre RGBD, on peut parfois choisir en paramètre l'algorithme pour générer notre Hash. Il ne faut plus l'utiliser.

SELECT SHA('Bonjour'); # Fonction qui génère un Hash de 160 bits (40 caractères qui sont généré a la fin). Un peu plus sécurisé que MD5 mais ce n'est plus utilise. L'appel de cette fonction produit toujours le même Hash. Pour la comparaison de Hash sur un système de mot de passe, d'authentification, il ne faut pas générer des Hash aléatoire a chaque appel de fonction avec le même texte en clair.

SELECT SHA1('Bonjour'); # Alias de SHA().

SELECT SHA2('Bonjour', 256 ); # Fonction qui permet de choisir un algorithme (sur 224 bits, 256 bits, 384 bits ou 512 bits) en paramètre pour générer un Hash. Plus le nombre de bits est important, plus le processus de Hashage serra long et plus la donnée sera sécurisée. Le temps en informatique s'exprime en seconde, voir en milliseconde.

#PostgreSQL (fonctions et paramètres en minuscule).

SELECT DIGEST('Bonjour'); # Fonction qui permet de créer une empreinte numérique. L'algorithme ('md5', 'sha1','sha224','sha256','sha384','sha512') qui se passe en paramètre s'indique sous forme de chaine. On préfèrera choisir le 256 ou le 512 (plus lent) que md5 ou sha1 qui ne sont plus utilises.

# Oracle Database (fonctions et paramètres en majuscule).

STANDARD\_HASH('Bonjour','MD5'); # Fonction qui permet de créer une empreinte numérique. Les algorithme en second paramètre sont : 'md5', 'sha1','sha224','sha256','sha384','sha512'.

# SQL server (avec l'extension TransactSQL).

HASHBYTES('SHA2\_256', 'Bonjour'); # Fonction qui permet de créer une empreinte numérique avec une chaine de caractères ou même des données binaires. Les algorithmes en premier paramètre sont : 'md2', 'md4','sha','sha1','sha2\_224','sha2\_256','sha2\_384','sha2\_512'. md2 et md4 sont des vieux algorithmes encore moins sécurisé.

# SQLite.

# None. RGBD light, pas de fonctions en natif pour le Hashage et le chiffrement.

#2# Fonctions de chiffrement.

# MySQL.

SELECT AES\_ENCRYPT('Bonjour', 'secret'); # Fonction de chiffrement. Le premier parametre est la donnees en clair et le second parametre est la clee de chiffrement (information secrete). La fonction retourne du binaire mais le terminal n'affiche pas du binaire. C'est le resultat du chiffrement de la chaine Bonjour.

SELECT AES\_DECRYPT('info\_chiffree'); # Fonction inverse qui utilise toujours l'algorithme AES.

SET @passw = AES\_ENCRYPT('Bonjour', 'secret');

SELECT @passw; # Afficher le contenu de la variable.

SET @text = AES\_DECRYPT(@passw, 'secret');

SELECT @text; # Afficher notre information en clair.

# Attention, une information chiffrée permet de retrouver la donnée en clair mais un Hash ne permet pas de revenir en arrière.

# Conclusion MySQL

SQL #22 - Conclusion MySQL

Formation très généraliste.

Un RGBD, MySQL dans un cadre privé ou professionnel.

On a abordé :

* L'installation et la configuration de MySQL.
* Les différents moteurs de stockage (très important).
* Les notions relatives aux bases de données.
* Les bases ;
* Les tables ;
* Les contraintes sur les différents champs de tables ;
* Les utilisateurs et les droits des utilisateurs ;
* Les requêtes usuels/standards (séance 9);
  + Lecture de données (SELECT);
  + La création de données ;
  + Mise à jour et suppression de données ;

=> appelé un CRUD (Create, Read, Update, Delete). CR = récupération de données, puis mettre a jour une base de données et supprimer des elements. Dans le web, la manipulation de données doit essayer de satisfaire ce principe de CRUD (jusqu'à la seance 12).

* Les fonctions d'agrégation ;
* Les fonctions natives de MySQL (min, max, nombre d'occurrences dans une table);
* Les groupements de résultats (Unions, Intersections, Jointures) -> a partir de la seance 12, on a vu comment récupérer les données d'un table et voir comment travailler avec plusieurs tables a la fois selon le principe de bases de données relationnelles.

Mise à disposition de fonctions par thèmes (chaines de caractères, heures et dates, Fonctions mathématiques et sécurité);

-> faire usage de la sécurité avec un langage de programmation si on fait autre chose que l'administration simple de base de données.

Source :

<https://sql.sh/>