***SQL :***

Les bases de données :

Leur intérêt premier est de stocker, mais elles sont aussi utiles pour extraire des informations au bon vouloir, trier des données, ect…

Les bases de données les plus connues sont :

***SGBDR  (système de gestion des bases de données relationnelles) :***

-DB2(IBM)  
-SQL Sewer (MS)

-Oracle DB (Oracle)

-MySQL (Sun Oracle) la plus utilisée

-Maria DB (fork de MySQL) deuxième plus utilisée

-Postgress (open source)

-Mongo DB.(base non SQL)

Les bases de données sont pensées comme des tables (de la même manière qu’un tableau excell).

Dans les bases SQL, toutes les lignes ont la même structure (un tableau ou chaque colonne définit l’information relative à la ligne sur un sujet donné).

L’intégration :

L’intégration permet d’avoir un seul fichier qui sert à plusieurs domaines pour avoir une plus grande flexibilité.

Par exemple, le service commercial et le service RH ont une base de données en commun afin de ne pas avoir à remplir deux fois les informations.

La sécurité :

Le dev de la BD doit gérer la base de données pour qu’elle soit la plus juste possible et doit avoir une tolérance aux erreurs.

Son code doit également empêcher les actes malveillants envers elle et donc avoir une tolérance aux actes malveillants.

Enfin, il faut faire attention aux pannes, ce qui est le job de l’admin réseau.

My SQL :

Base de données dominante (100% d’utilisation en France, plus de 95% dans le monde) grâce à sa licence très permissive au niveau commerciale notamment, cette base de données est incontournable pour tout développeur web qui se respecte.

Le chemin pour accéder aux commandes de la BD (utiliser la console pour travailler) :

Dans le fichier de wamp, il faut aller dans :

Bin/

Database/

MySQL-x.y.z/

Bin/

Ensuite faire shift+clic droit => ouvrir une fenêtre de commandes ici.

Quand on rentre une commande, elle doit ***IMPERATIVEMENT*** se terminer par un ***;***

Pour se connecter à la BD via la console  il faut taper :

***mysql –u root(le nom d’utilisateur) –p***

il faut ensuite rentrer son mdp (pour nous, root)

pour vérifier la version de mysql il faut taper :

***show version() ;***

Pour voir les différentes bases de données il faut taper :

***Show databases ;***

Pour créer une nouvelle bd :

***Create database nomdelabd ;***

Pour accéder à une base de données en particulier :

***use nomdeladb ;***

Pour voir ce qu’elle contient :

***show tables ;***

Pour supprimer une base de données :

***drop database nomedeladb ;***

Pour supprimer une table d’une DB :

***drop table nomdetable ;***

Pour créer une table :

***Create table nomdetable(nomdecollone1 type attribut, nomdecolonne2 type attribut, …) ;***

Pour voir une table :

***describe nomdetable ;***

Pour vider une table sans la supprimer :

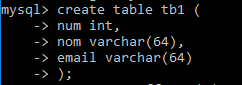
***truncate table nomdetable ;***

Les types les plus utilisés :

***Int :*** permet de rentrer des valeurs numériques.

***Char(x)*** : permet de rentrer x caractères alpha numériques. Par exemple pour un char(4), si on ne met qu’une lettre, les trois espaces restants seront comblés par des espaces. C’est plus utile pour les valeurs fixes, comme les années qui prendront toujours 4 espaces ou les codes postaux qui en prennent toujours 5. ***Le x peut aller jusqu’à 255 maximum.***

***Varchar(x) :*** Contrairement au char(x), ici le reste ne sera pas comblé par des espaces. On peut utiliser x nombre de caractères au maximum. Si on met varchar(5) et qu’on ne met qu’une lettre, alors le serveur dira qu’il n’y a qu’un espace d’utilisé, ce qui permet d’économiser de la place dans les données. ***Le x peut aller jusqu’à 65 000.***



On ne met pas de virgule après le dernier élément créé, sinon la console va s’attendre à ce qu’on veuille en rajouter un. Ici, il n’y a pas de virgule après le mail.

Les attributs :

Les attributs sont des facteurs qui vont être appliqués à des données entrées dans les tables. Si on ne précise rien, la donnée est null par défaut.

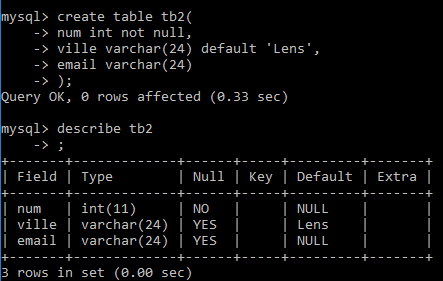
Les attributs les plus utilisés sont :

***not null*** (il faut nécessairement rentrer quelque chose)

***default***  a moins d’être modifiée, la valeur donnée sera la valeur par défaut.

***primary key*** (est une sorte d’ID qui permet de distinguer une ligne de manière unique.)

***unique*** (Utile par exemple pour les numéros de factures).



Insérer les infos sur une table :

Avant tout, vérifier la db sur laquelle on travaille avec :

***Select database() ;***

Ensuite il faut taper :

***Insert into nomdetable(colonnex,colonnex,… selon ce qu’on veut insérer dans telle colonne)***

***Values(val1,val2,valx…) ;***

L’ordre est très important parce que la value1 ira se placer dans la première colonne appelée.

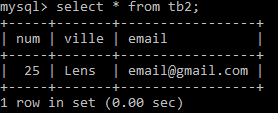


**Il est possible de ne pas préciser les colonnes. Mais dans ce cas elles seront toutes visées par défaut et il faudra donc intégrer *TOUTES* les informations.**

Voir ce qu’on a entré comme informations :

*Pour voir l’intégralité de ce qu’on a entré :*

***Select \* from nomdetable ;***

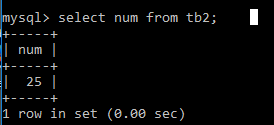


Ici, Lens était la valeur par défaut, donc nous n’avons pas eu besoin de la modifier.

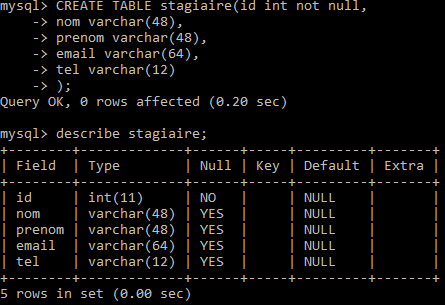
Pour voir juste une colonne en particulier, ou plusieurs mais pas tout :

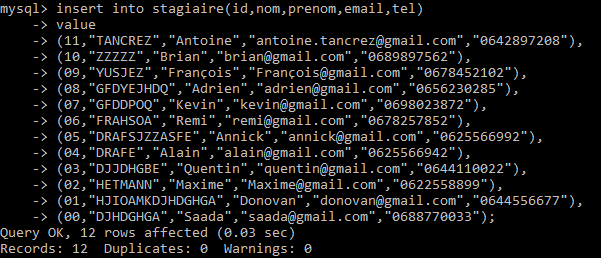
*Select colonne from nomdetable* pour une colonne unique.

***Select col1,col2,colx from nomdetable*** pour plusieurs colonnes (il faut les séparer d’une virgule.

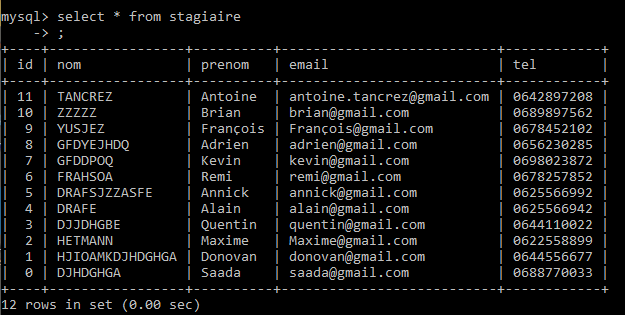


Un exemple de « vraie » base de donnée créée :





On peut intégrer plusieurs lignes en même temps, il suffit de séparer les différentes values par une virgule et mettre un point-virgule à la dernière.



Pour importer une base de donnée :

On peut taper la base de données sur un fichier texte exactement de la même manière que ce qu’on aurait tapé sur la console.

Sur la console on rentre :

***source sourcedufichier ;***

***Techniquement, on peut tout taper sur un éditeur de texte et l’importer sur la console, donc on peut même créer des bases de données, tables ect.***

Il faut s’identifier avant d’importer le fichier.

Modifier les tables :

Ajouter une colonne :

***Alter table nomtable add colx typex ;***

Supprimer une colonne :

***Alter table nomtable drop colx ;***

Ou

***Drop column colx ;***

Modifier une colonne :

Alter table nomtable modify colx typex ;

Changer le nom d’une colonne :

***Alter table nomtable change anciennom nouveaunom typex ;***

Pour écrire un commentaire :

On peut faire --text en commentaire et le commentaire fera toute la ligne.

On peut aussi utiliser dièse ou double dièse.

La meilleure solution reste de faire ***/\* commentaire \*/***

Trier les informations :

On peut comme précisé plus haut afficher tout ou partie d’un tableau :

***Select column from table ;***

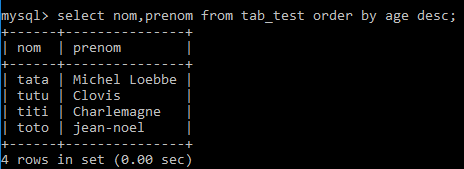
Mais on peut les trier par rapport à une colonne en particulier (même si celle-ci n’est pas affichée dans la sélection :

***Select colx from table order by coly ;*** on trie la colonne x par rapport aux données de la colonne y.

Si on ne précise pas, le trie se fera dans l’ordre croissant des valeurs.

Pour trier dans l’ordre inverse, il faut mettre :

*Select colx from table order by coly****desc ;*** à la fin de la ligne de commande.



Pour regarder une base de données soit sur un nombre N ou M lignes à partir d’un nombre N :

***Select \* from table limit N ;*** Ici, on prendra les N premières lignes.

***Select \* from table limit N,M ;*** Ici, on affichera a partir de la ligne N M lignes.

Extraire d’un tableau selon une donnée en particulier dans une colonne :

***Select \* from table where colonne = « mot ou chiffre recherché » ;***

On peut utiliser les signes < et >pour les chiffres.

Il est possible de faire :

***Select \* from table where colonne like « %donnée% » ;***

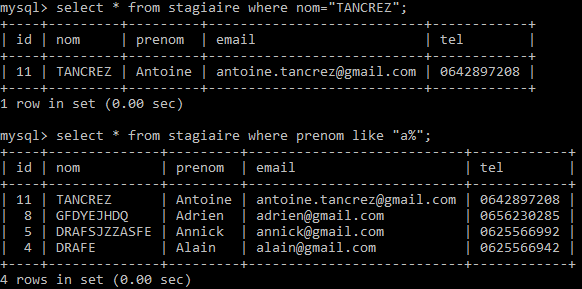
Le like permet d’afficher tout ce qui contient ce qu’il y a dans les % même si ce n’est qu’une partie du mot qui sera retourné.

Sur MySQL :

Quand on met ***like donnée***, le like vaudra le = . Dans la pratique, ce n’est pas utilisé car le = le fait déjà.

Si on met ***like donnée%*** alors ça affichera tout ce qui commence par la donnée.

Si on met ***like %donnée***, alors ça affichera tout ce qui finira par la donnée.



Si on veut par exemple les données qui commencent par un a et qui contiennent un b dans le mot alors il faut mettre :

***Like « a%b% » ;***

Si on veut par exemple que les prénoms qui commencent par un a et qui ont un t en troisième position :

***Like « a\_t% » ;*** chaque caractère vide sera remplacé par un underscore.

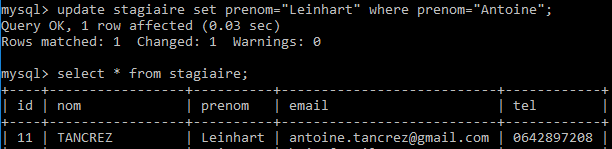
Mettre à jour des données :

Si on veut mettre à jour toutes les données d’une colonne pour donner la même donnée :

***Update nomtable set colonne = « donnée » ;***

Si on veut modifier les données d’une ligne en particulier ou alors cibler plus en détails :

***Update nomtable set colonne = « nouvelle donnée » where colonne= «donnée »  ;***



Imaginons qu’on ait une liste de numéros de téléphones.

Si on veut update en remplaçant les lignes sans numéros par « non communiqué » , il faut mettre :

***Update nomtable set tel = « non communiqué » where tel= « » ;***

Ici, les espaces vides sont ciblés par les guillemets vides.

Delete des données :

On utilisera toujours une condition pour ne pas supprimer toute une colonne. Il faut faire attention car

***Delete from table ;*** fonctionne et supprime tout et videra le tableau.

En général on mettra :

Delete from table where colonne= « donnée » ;

Par exemple, si on veut enlever les lignes qui n’ont pas de numéro de téléphone on mettra :

***Delete from table where tel= « » ;***

Si on veut sélectionner parmi un critère ou un autre on peut faire :

***Select\* from table where colonne in (« data1 »,« data2 »,…) ;***

Ici, ça sera les données qui sont soit le data1 ou le data2.

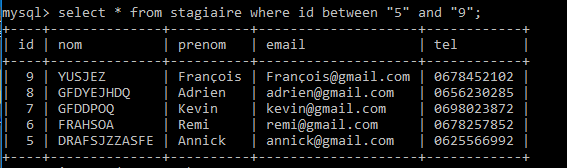
***Select\* from table where colonne not in (« data1 »,« data2 »,…) ;***

Si on veut sélectionner tout ce qui ne sera pas le data1 ou le data2.

On peut utiliser le mot between pour chercher entre deux valeurs ordonnées :

***Select\* from table where age between « 18 » and « 50 » ;***

Ici, on va sélectionner tout ce qui est entre 18 et 50 compris.



Les concaténations éphémères :

Elles servent par exemple à regrouper deux colonnes pour voir les données affichées ensemble (uniquement pour la requête).

***Select Concat(col,col,col) from table ;***

Pour afficher les deux colonnes ensemble.

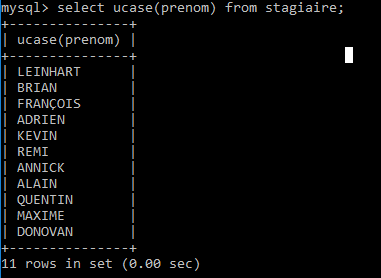
On peut renommer cette colonne éphémère avec un alias :

***Select Concat(col,col,col) as « nom » from table;***

On peut afficher les résultats d’une colonne tout en majuscule ou tout en minuscule :

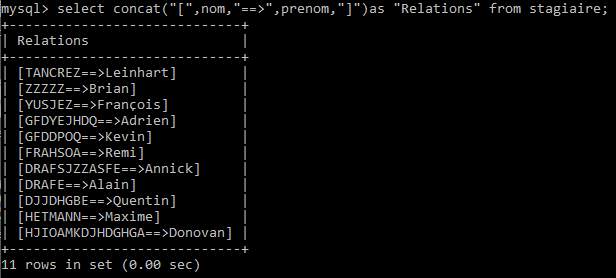
***Select ucase(col) from table ;*** Pour les majuscules.

***Select lcase(col) from table ;*** Pour les minuscules.



Il est possible d’ajouter dans la concaténation des valeurs qui ne sont pas des colonnes. Il suffit de les mettre entre guillemets.

Aussi, l’alias « as » peut être utilisé tout le temps.



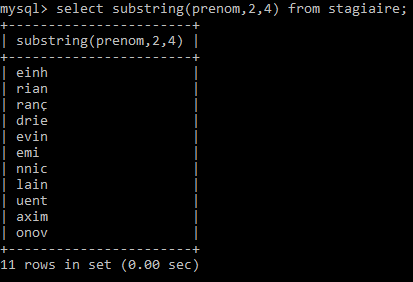
Ici, la colonne éphémère Relations contient les deux colonnes nom et prenom qui sont séparées par des caractères ajoutés pour l’affichage.

Afficher seulement une partie d’une colonne :

Il est possible d’afficher seulement une partie d’une colonne :

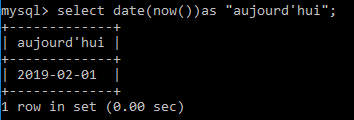
***Select substring(col,1,5) from table ;***

Ici, le premier chiffre détermine à partir de quel numéro de caractère on va afficher et le deuxième de combien de caractères on va afficher :



Afficher la date :

***Select date(now()) ;***



On peut modifier l’affichage :

***Select date\_format(now(), »%y-%M-%d %h-%i-%s ») as « today » ;***

***%y*** signifie l’année

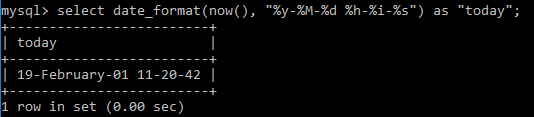
***%m*** signifie le mois. ***%M*** l’affiche en lettres.

***%d*** signifie le jour ***%W*** pour avoir le jour en lettres.

***%h*** signifie l’heure

***%i*** signifie les minutes

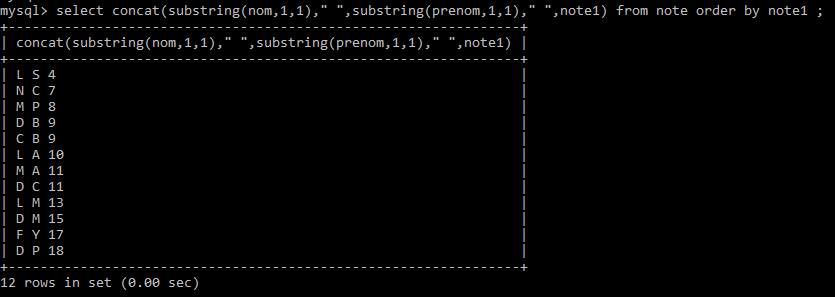
***%s*** signifie les secondes



Afficher le nombre d’éléments d’une colonne :

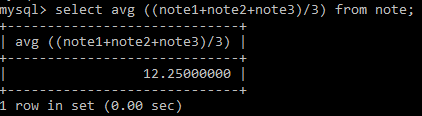
Pour afficher combien de lignes contient une colonne :

***Select count(col) from table ;***

******

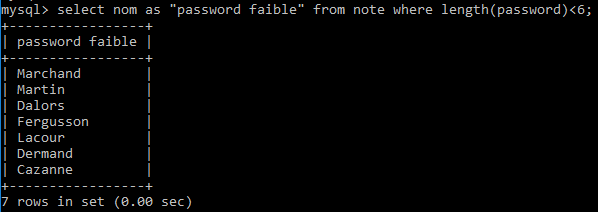
Ici, on a mis des substrings dans le concat pour retirer que les initiales.

Calculer une moyenne :

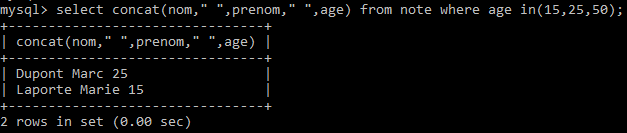


***Select avg (calcul de la moyenne d’une personne) from table ;***

Sélectionner selon une longueur :



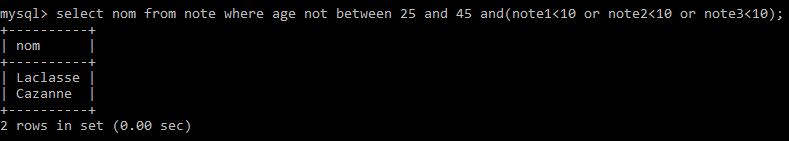
Utiliser le mot in pour trier :



Le mot in est pratique pour sélectionner les éléments qui font partie de telle, telle ou telle condition. Au lieu de mettre plusieurs or d’affilé, ici il suffit juste de faire une liste des éléments que l’on veut voir rentrer en tant que condition.

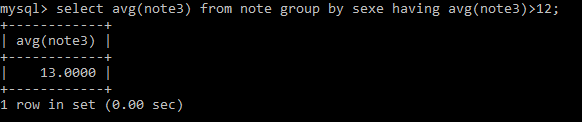
Le mot between ou not between :

En utilisant le mot between, on peut sélectionner tout ce qui est entre telle et telle valeur. Le not between permet d’exclure tout ce qu’il y a entre les valeurs :



Le mot having :

La condition HAVING en SQL est presque similaire à WHERE à la seule différence que HAVING permet de filtrer en utilisant des fonctions telles que SUM(), COUNT(), AVG(), MIN() ou MAX().



Datediff :

Dans le langage SQL, la fonction ***DATEDIFF()*** permet de déterminer l’intervalle entre 2 dates spécifiées. La fonction est utilisée avec les systèmes MySQL et SQL Server, mais s’utilise différemment :

MySQL : la fonction prend 2 dates en paramètres et retourne le nombre de jours entre les 2 dates

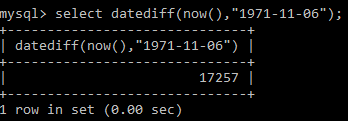
SQL Server : la fonction prend 2 dates ainsi qu’un paramètre déterminant sous quel intervalle de distance temporelle le résultat doit être retourné (nombre de jours entre 2 dates, nombre d’années, nombre de semaines, nombre d’heures …)

MySQL :

Sous MySQL, la fonction ***DATEDIFF()*** s’utilise dans une requête avec le format suivant :

***SELECT DATEDIFF( date1, date2 );***

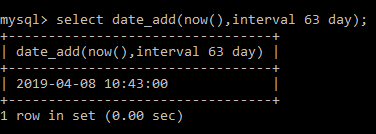
Les dates doivent être au format DATE (cf. AAAA-MM-JJ) ou DATETIME (cf. AAAA-MM-JJ HH:MM:SS). ***Pour que le résultat soit positif il faut que la date1 soit plus récente que la date2.***



Obtenir une date future ou passée en interval de jours :

***SELECT DATE\_ADD(NOW(),INTERVAL 5 DAY); # le futur***

***SELECT DATE\_SUB(NOW(), INTERVAL 5 DAY); # le passé***



Trier par rapport à une date de naissance :

-- Question 5:

select \* from soldats WHERE timestampdiff(year,date\_naissance,now())>25;

-- Cette requête permet de trier les soldats par rapport à leur date de naissance pour afficher

-- uniquement ceux qui ont plus de 25 ans.

--le year permet de tester en années. Le premier paramètre définit la date que l’on veut tester et le deuxième est ceux par rapport à quoi on va tester. On veut que ce résultat dépasse les 25 ans.

Mettre en majuscule de façon permanente :

-- Question 7:

update soldats

set nom = ucase(nom);

-- Permet de modifier les noms de familles pour les mettre en majuscule de façon permanente.

Le mot limit :

-- Question 8:

select \* from soldats order by date\_naissance limit 2;

-- Permet d'afficher uniquement les deux soldats les plus âgés.

Delete seulement une partie d’une table :

-- Question 8:

delete from soldats order by date\_naissance limit 2;

-- Cette ligne permet de delete uniquement les deux plus vieux soldats.

Réaliser un update mais partiel :

Il est possible de réaliser un update que de certaines données dans une colonne :

-- Question 10:

update soldats

set grade = grade + 1 where grade <5;

-- Cette commande permet d'update le grade de tout le monde sauf ceux qui ont un grade égal à 5.

-- Question 11:

update soldats

set grade = grade - 1 where grade =3 and sexe="h";

-- Ici, on baisse de 1 le grade uniquement des hommes qui ont un grade = 3.

update soldats

set date\_naissance= date\_add(date\_naissance,interval 1 day) where sexe="f" and grade =4;

-- Ici, on ajoute un jour à la date de naissance uniquement aux femmes qui ont

-- un grade = 4.

Arrondir à un certain nombre de chiffres après la virgule :

On utilisera la commande round :

select round(avg(satellite),2) from planete\_antoine;

Le chiffre après la virgule dans la commande définit combien de chiffres on veut afficher après la virgule.

Créer une colonne qui affiche un chiffre en décimal :

Le int affiche des chiffres mais uniquement entiers. Pour les décimaux, on utilisera decimal :

nomdecolonne decimal(16.2) not null,

Le premier numéro va définir le nombre de caractères max affichés (virgule comprise), le deuxième le nombre de chiffres après la virgule.

Requête dans une requête avec le mot in :

Il est possible d’imbriquer des requêtes pour avoir des résultats plus précis :

select planete from planete\_antoine where satellite in (select max(satellite)from planete\_antoine);

Le résultat retourné après le in sera retourné dans la requête principale.

On peut imbriquer les requêtes sans le mot in. Les parenthèses sont le plus importantes. Le mot in est important quand on recherche un résultat non fixe.

Afficher un jour précis de manière simple :

select dayname("2018-12-23") as "veille du réveillon";

Le jour va s’afficher avec le jour en lettres. C’est la même chose que %w.

Les primary keys contenant plusieurs champs :

create table planete\_antoine(

Planete VARCHAR(15) NOT NULL,

Distance int NOT NULL,

Rotation VARCHAR(5) NOT NULL,

Revolution VARCHAR(30) NOT NULL,

Satellite int NOT NULL,

PRIMARY KEY(planete,distance,rotation)

);

Il faut créer les colonnes de manière classique sans préciser primary key, et créer la primary key à la fin.

Select plusieurs tables en même temps :

Il faut utiliser le mot union qui va permettre de lier les deux tables. Evidemment, il faut que les champs des deux tables soient identiques (mêmes colonnes, ect).

select \* from table1

UNION

select \* from table2;

Créer une seconde table qui a les mêmes propriétés que la première :

Si on veut créer une table identique à une table déjà créée, il existe un moyen rapide évitant de tout retaper et ainsi permettant d’éviter les erreurs humaines :

Create table table2 like table1;

Enregistrer une requête pour éviter de la retaper :

Il est possible de créer une requête qui fonctionne comme un alias afin d’éviter de retaper toujours la même ligne grâce à la commande ***create view.***

Ce qui est créé est considéré comme une table virtuelle. C’est-à-dire qu’elle apparaît comme une table, on peut la manipuler comme une table mais ce n’est pas une table. Elle n’est qu’une table par sa formule. Elle n’existe pas réellement et est juste créée à la demande.

-- Ici, le as sert à introduire la requête pour dire cette view signifie la requête entre parenthèses.

create view nomdeview as (select \* from planete\_antoine where satellite>20);

Cette formule à pas mal d’utilités, notamment limiter l’accès à uniquement certaines infos pré-triées.

Pour voir quelle formule a été utilisée pour créer une view :

show create view test\_view;

La formule n’est pas très standard et l’affichage qui va apparaître est normal.

Les foreign keys :

Une foreign key sert à mettre deux tables en relation. Si on doit mettre les données d’une colonne en relation avec les données d’un autre tableau, il suffira de faire un foreign key qui fera référence à une colonne donnée d’un autre tableau.

-- Par exemple: table 1:

create table1(

nom VARCHAR(15) NOT NULL,

prenom varchar(15) NOT NULL,

age VARCHAR(3) NOT NULL,

services int not null,

FOREIGN KEY (services) REFERENCES table2(services\_nom)

);

-- La table 2:

create table2(

services\_nom int,

compagny varchar(30) not null

);

-- Ici, la colonne services de la table 1 doit être la même que service ID de la table 2 Elle peut être nulle.

-- Elle ne peut pas être différente.

-- Dans la table 1, la colonne services est liée à services\_id de la colonne 2.

Les jointures :

On peut faire des recherches en corrélant plusieurs tables pour avoir une recherche qui va afficher les résultats provenant des deux tables dans une seule :

select nom,prenom,nom\_services

FROM

table1 join table2

on services = nom\_services

;

-- Ici, nom et prenom proviennent de la table 1. nom\_services provient de la table2.

-- On joint les deux tables grâce à la foreign key où le numéro de service est égal à l'id du nom\_services.

Il est possible d’ajouter le mot ***left*** pour prendre en compte les résultats où il n’y a pas de réponse mais un champ nul ou pas de lien dans la deuxième table :

select nom,prenom,nom\_services

FROM

table1 left join table1

on services = nom\_services

;

Il est possible d’utiliser le mot ***right***. Dans ce cas, ce sont les données de la première table qui sont nulles dans la deuxième.

Afficher toutes les données d’une colonne sans les redondances :

Il existe une commande pour tout afficher d’une colonne en ne prenant pas en compte les doublons :

select DISTINCT numclient from commande;

-- Cette commande permet d'afficher toutes les données d'une colonne sans afficher les doublons. C'est très pratique

-- Pour voir par exemple combien de clients ont commandé, en ne les affichant qu'une seule fois s'ils ont effectué

-- plusieurs commandes.

select count(distinct(numvendeur)) from commande having count(numcmde)>=1;

-- Ici, on vient compter les vendeurs qui ont fait au moins une commande. Il faut donc les distinguer pour

-- éviter les doublons.

Pour faire une recherche de quelque chose de null :

Dans MySQL, donnée=null ne fonctionne pas. Il faut mettre ***donnee is null***:

select nom,prenom,numservice from filialea

where numservice is null

UNION

select nom,prenom,numservice from filialeb

where numservice is null;

-- Ici, le is null signifie que l'on va rechercher uniquement les lignes qui ont la colonne

-- numservice de null.

--Evidemment, on fait deux fois la requête parce qu’on utilise le union pour lier deux tables qui ont des colonnes identiques.

Lier une table à plusieurs foreign keys :

select concat(substring(prenom,1,1),".",nom),nom\_service,matricule

from

filialea join voitures

on filialea.numvoiture=voitures.idvoiture

join services

on filialea.numservice=services.idservice

UNION

select concat(substring(prenom,1,1),".",nom),nom\_service,matricule

from

filialeb join voitures

on filialeb.numvoiture=voitures.idvoiture

join services

on filialeb.numservice=services.idservice

;

-- Ici, on utilise deux clefs étrangères. Pour ce faire, on va join la table à la première qui a la foreign key

-- préciser ce qui les relie, et faire la même chose sur la deuxième table possédant la foreign key. Union permet de relier à une autre table identique

-- à la première , donc on peut comparer les deux tables.

Faire une recherche grâce à une requête imbriquée :

select nom\_service,count(numservice)

from

(select nom,prenom,nom\_service,numservice

from

filialea join services

on filialea.numservice = services.idservice

union

select nom,prenom,nom\_service,numservice

from

filialeb join services

on filialeb.numservice = services.idservice)

as totaux

group by numservice

;

-- Ici, l'idée est de compter les personnes qui appartiennent à un service d'une entreprise.

-- On va donc faire deux colonnes, la première pour avoir le numéro de service et la seconde

-- va afficher le nombre de personnes qui y travaillent.

-- Le from englobe l'union des deux tableaux qui vont chercher les informations requises grâce

-- à leur foreign key. Leur select doit nécessairement appeler les colonnes du select principal

-- pour avoir le bon résultat.

-- Le as est obligatoire. Pour un gros from entre parenthèses comme celui-ci, SQL demande de lui

-- donner un alias.

-- On trie par numéro de service pour afficher chaque service.

Un exemple de join entre plusieurs tables :

-- Cette fonction met en relation toutes les tables qui concernent le livre pour une barre de recherche:

-- Titre vient de la table livre, nom vient de la table auteur, genre vient de la table categorie et

-- motcle vient de la table motclef

select titre,nom,genre,motcle

from

-- Pour commencer, on va join le livre à son ou ses auteurs:

ecritpar join livre

on id\_livre = livre.id

join auteur

on id\_auteur = auteur.id

-- Ensuite on join le livre à sa catégorie:

join categorie

on livre.rayon=categorie.id

-- Enfin, on relie les livres à leurs mots-clefs:

join motcleflivre

on motcleflivre.id\_livre=livre.id

join motclef

on ID\_MOTCLEF=motclef.id

-- On trie par auteur pour avoir leurs différentes oeuvres:

order by nom

;