SQL

Qu'est ce que SQL ?
Qu'est ce qu'un SGBD ?
Qu'est ce que MySQL ?
Téléchargement et installation de Wamp 3.2
Création de la base villes ➔creation des table selon le schema suivant
NB : il faut faire attention à l'encodage de la base et au moteur utilisé, on privilégie l'encodage UTF-8 et le moteur InnoDB.
NB : ce schema n'est pas 100% optimisé pour des raisons pédagogiques.
Region : id, nom, population, superficie, densite
Departement : id, code, region_id (#region), nom, population, superficie, densite
Responsabilite : id, role
Responsable : id, role_id (#responsabilite), nom, prenom, dat_naiss, dat_deb, dat_fin
Ville : id, nom, depart_id (#departement), maire_id (#responsable), cod_post, population, superficie, densite
Explication du principe des clés primaire et étrangère
Définition des clés étrangères : aller à la page de la table ➔ structure ➔ vue relationnelle
Explication du principe de l'intégrité référencielle
Lors de la definition des clés étrangères, on nous demande de definir deux regles :

- **ON DELETE** : Comportement que devra avoir le SGBD si vous supprimez un enregistrement qui est référencé dans une autre table.

Par exemple, que faire si dans la table *region* je supprime une ligne qui est référencée par un ou plusieurs enregistrements de la table *departement* ?

- **ON UPDATE** : Même chose mais dans le cas de la mise à jour de l'enregistrement qui est référencé.

Chacune des regles nous demande de choisir parmi 4 choix possibles :

- **RESTRICT** OU **NO ACTION*** : Ne va rien faire. Par exemple : si je supprime un enregistrement référencé, et bien les enregistrements qui le référençaient vont dorénavant référencer un enregistrement qui n'existe plus. Rarement une bonne idée. Si il s'agit d'une mise à jour : ont met à jour toutes les clefs étrangères concernées, si cela est applicable.

NB : Cette équivalence entre RESTRICT et NO ACTION est propre à MySQL. Donc attention si vous utilisez un autre SGBD!

Exp : si je supprime la région hauts de France qui a l'id 2, les départements qui la référencent vont toujours avoir region_id 2

- **SET NULL** : La clef étrangère reçoit la valeur NULL. Tous les enregistrements qui référencent celui qui a été modifié vont recevoir la valeur null. Peut être utile dans le cas d'un DELETE.

Exp : region_id va avoir null

- **CASCADE** : Mise à jour en cascade. Celui-là, c'est la violence. Il mettra à jour / supprimera automatiquement les enregistrements qui référencent l'enregistrement qui a été modifié / supprimé.

Si je modifie l'id de la region, region_id des departements va etre modifiée, et si je la supprime (la region) tous les departments qui y sont rattachés vont etre supprimés.

Departement : region_id ➔ region [delete : restrict // update : cascade]
Responsable : role_id ➔ responsabilite [delete : restrict // update : cascade]
Ville : depart_id ➔ departement [delete : restrict // update : cascade]
maire_id ➔ responsable [delete : restrict // update : cascade]

La commande de sélection

La commande SELECT : permet de selectionner des enregistrements d'une ou plusieurs tables.

```
SELECT * FROM ville;
SELECT nom, cod_post FROM ville
```

La variante distinct permet de n'avoir que des valeurs sans doublons dans le but l'éliminer la redondance des données

```
SELECT DISTINCT nom FROM citoyen
```

NB : le mot clé distinct ne s'applique qu'à une colonne (ne s'applique qu'à la colonne nom dans cet exemple)

Alias sur les colonnes

L'alias est tres pratique dans le cas des noms de colonnes longs ou qui posent des difficultés d'identification.

Renommer le nom de la colonne

```
SELECT population AS pop FROM ville
```

Il est tout de meme possible de renommer le nom de la colonne et de la table

```
SELECT population AS pop
FROM ville AS v
where v.population > 10000
```

La clause where

Permet de selectionner des enregistrements avec des conditions

```
SELECT *
FROM ville
WHERE densite >= 1000
```

Ex 1: sélectionner les villes dont la superficie dépasse les 10 Km2

```
SELECT *
FROM ville
WHERE superficie > 10
```

Ex 2 : sélectionner le nom, la population et la densité des villes dont la population dépasse 100 000 et la densité 1000 habitants.

```
SELECT nom, population, densite
FROM ville
WHERE population > 100000 AND densite > 1000
```

Ex 3 : chercher les villes dont la population depasse 500 000 habitants ou la superficie depasse 7 km2

```
SELECT *
FROM ville
WHERE population > 500000 OR superficie > 7
```

La clause IN

S'utilise avec WHERE pour tester si une colonne a une valeur contenue dans une plage de données, elle est tres pratique pour etre utilisée à la place de plusieurs operateurs OR.

NB : Cette syntaxe peut etre associee a l'operateur NOT pour recherche toutes les lignes qui ne sont pas egales a l'une des valeurs stipulées.

Exp : selectionner les noms des departements des regions ile de France et hauts de France.

```sql
SELECT * FROM departement
WHERE region_id IN (1,2)
```

Ex 1 : determiner les villes sont le code postal correspond à l'un des suivants : 62100, 62500 ou 62000

```sql
SELECT *  FROM ville
WHERE cod_post IN (62100, 62500, 62000)
```

ex 2 sélectionner les départements qui appartiennent à la normandie ou au grand est

```sql
SELECT nom
FROM departement
WHERE region_id IN (3,4)
```

Ex 3 : determiner les villes dont le code postal NE correspond PAS à l'un des suivants : 62300, 62500 ou 62400

```sql
SELECT * FROM ville
WHERE cod_post NOT IN (62300, 62500, 62400)
```

La clause BETWEEN

Elle s'utilise avec where et elle sert à sélectionner un intervalle de données, le plus souvent des entiers ou des dates.

Exp : selectionner les villes dont la population est entre 50000 et 100000 habitants.

```sql
SELECT * FROM ville
WHERE population BETWEEN 50000 and 100000
```

Ex 1 : selectionner les responsables qui sont nés entre 1970 et 1985

```sql
SELECT * FROM responsable
WHERE dat_naiss BETWEEN '1970-01-01' AND '1985-12-31'
```

2eme solution

```sql
SELECT *
FROM responsable
WHERE year(dat_naiss) BETWEEN 1970 and 1985
```

Utilisation de fonctions avec where

On peut utiliser des fonctions dans la clause where pour extraire des informations des colonnes.

Exp : afficher les responsables dont la date de naissance correspond à un lundi

```sql
SELECT *
FROM responsable
WHERE DAYNAME(dat_naiss) = 'monday'
```

Ex : afficher l'id, le nom complet et la date de naissance du responsable "Dumont Yvonne"

```sql
SELECT  id, concat(nom, " ", prenom) AS nom_complet, dat_naiss
FROM responsable
WHERE concat(nom, " ", prenom) = 'Dumont Yvonne'
```

Liste complete des fonctions MySQL
https://www.w3schools.com/mysql/mysql_ref_functions.asp

La clause LIKE

Elle s'utilise avec where et permet d'effectuer des recherches en fonction d'un modele.

Dans le modele, on utilise les symboles suivants % (un ou plusieurs caractères) et _ (un seul caractère)

NB : like est insensible à la casse

Exp : selectionner les responsables dont le prenom commence par F

```
SELECT *  FROM responsable
WHERE prenom LIKE 'f%'
```

Ex 1 : selectionner les responsables dont le nom commence par R et finit par t

```
SELECT *  FROM responsable
WHERE nom LIKE 'r%t'
```

Ex 2 : chercher les citoyens dont le prenom ne contient que deux lettres.

```
SELECT * FROM citoyen
WHERE prenom like '__'
```

Ex 3 : déterminer les noms des responsables dont le nom complet commence par D et finit par s

```
SELECT id, concat (nom, prenom) as nom_complet
FROM `responsable`
WHERE concat (nom, prenom) like 'D%s'
```

Ex 4 : chercher les citoyens dont le prenom ne contient pas plus que deux lettres.

```
SELECT *
FROM citoyen
WHERE CHAR_LENGTH (prenom) <= 2
```

La clause IS NULL / IS NOT NULL

Sélectionner les resultats dont la valeur d'une colonne donnée est nulle (is null) ou pas nulle (is not null).

Exp : déterminer les responsables qui ne sont affectés à aucun poste.

```
SELECT *  FROM responsable
WHERE role_id IS NULL
```

Ex : déterminer les responsables qui sont affectés à une responsabilité.

```
SELECT *  FROM responsable
WHERE role_id IS NOT NULL
```

La clause ORDER BY

Permet de faire un tri des données sur une ou plusieurs colonnes selon l'ordre ascendant ou descendant.

NB : par défaut les données sont classées par ordre ascendant.

Exp : chercher toutes les villes, les ordonner selon la densité.

```
SELECT *
FROM ville
ORDER BY densite
```

Ex : classer les departements selon la population par ordre descroissant

```
SELECT *
FROM departement
ORDER BY population DESC
```

NB : il est possible de faire le tri sur plusieurs colonnes et dans l'ordre ascendant ou descendant pour chacune.

```
SELECT *
FROM responsable
ORDER BY dat_naiss DESC, dat_fin DESC
```

La clause LIMIT

Elle sert à faire une limitation sur les résultats retournés par la requete.

Exp 1 : afficher les 3 plus petits departements

```
SELECT * FROM departement
ORDER BY superficie
LIMIT 3
```

Ex : afficher les deux plus grandes regions en fonction de la densité

```
SELECT *
FROM region
ORDER BY densite DESC
LIMIT 2
```

La clause LIMIT peut etre utilisée avec un declage de lignes.

Exp 2 : afficher les 3 departements qui viennent apres la position 5 selon le code postal

```
SELECT *
FROM departement
ORDER BY code
LIMIT 5, 3
```

➔ prendre 3 enregistrements apres un decalage de 5.

La meme requete peut etre ecrite d'une façon plus explicite et c'est celle qui est recommandée

```
SELECT * FROM departement
ORDER BY code
LIMIT 3 OFFSET 5
```

ex : chercher les 10 departements les plus denses et qui viennent apres la position 20

```
SELECT *
FROM departement
ORDER BY densite DESC
LIMIT 10 OFFSET 20
```

NB : dans l'exemple de notre base, on a 24 enregistrements dans la table departement
si on fait un decalage de 20, il ne reste que 4, alors qu'on demande 10 enregistrements
dans ce cas il ne prend que les 4 existants
si jamais on avait plus que 10 apres un decalage de 20, il aurait pris seulement les 10 premiers

Les fonctions

Pendant ce cours on va se concentrer sur les fonctions d'agrégation statistiques, vous trouverez plus loin un lien vers toutes les fonctions MySQL.

NB : les fonctions d'agrégation sont souvent utilisées avec la clause GROUP BY surtout dans les cas des jointures (à voir plus tard), car si cette dernière est omise, la requete ne retourne qu'un seul résultat.

AVG() : permet de calculer la moyenne d'un ensemble de valeurs numériques.

Exp : calculer la moyenne de la population des departements

```
SELECT AVG(population)AS moyenne_pop
FROM departement
```

NB : dans cet exemple, on ne va pas utiliser GROUP BY parce qu'on ne va pas faire de jointures.

ex 1 : calculer la moyenne de la superficie des departements

```sql
SELECT AVG(superficie) AS moyenne_sup
FROM departement
```

ex 2 : calculer la densité moyenne des regions

```sql
SELECT AVG(densite) as moyenne_densite
FROM region
```

Ex 3 : calculer la population moyenne des regions à l'exception du grand est

```sql
SELECT AVG (population)
FROM region
where nom != 'grand est'
```

ex 4 : calculer la moyenne de la superficie des departements des regions normandie et grand est

```sql
SELECT AVG(superficie) AS moyenne_sup
FROM departement
WHERE region_id IN (3,4)
```

Ex 5 : calculer l'age moyen des responsables

```sql
SELECT AVG(   YEAR(CURDATE()) -    YEAR(dat_naiss)   )
from responsable
```

COUNT() : elle permet de connaitre le nombre de lignes

```sql
SELECT count(*) AS nb_citoyens
FROM citoyen
```

NB : à la place de l'etoile, on peut mettre le nom de n'importe quelle colonne, ca ne change rien, puisque les colonnes n'ont pas d'importance, ce qui importe ici est le nombre de lignes.

NB : count() peut etre aussi utilisée avec distinct pour compter le nombre distinct des enregistrements, par contre dans ce cas le nom de la colonne est primordial.

```sql
SELECT count(DISTINCT nom)
FROM citoyen
```

Ex 1 : compter le nombre de responsables

```sql
SELECT COUNT (*) AS nb_resp
FROM responsable
```

Ex 2 : compter le nombre de departements dont le code postal est < 90

```sql
SELECT COUNT(*) AS nb_dep
FROM departement
WHERE code < 90
```

MAX() : elle retourne la valeur maximale d'une colonne

```sql
SELECT MAX(population)
FROM departement
```

ex : calculer la densité maximale des régions

```sql
SELECT  MAX(densite) AS max_ densite
FROM region
```

MIN() : meme chose que max mais elle retourne la valeur minimale.

SUM() : retourne la somme des valeurs numériques d'une colonne

```sql
SELECT  SUM(population) AS somme_pop_depart
```

```
FROM departement
```

Ex : calculer la superficie totale des regions IDF et du grand est

```
SELECT SUM(superficie) AS densite_region
FROM region
WHERE id IN (1, 4)
```

Lien vers la liste complète des fonctions MySQL
https://www.w3schools.com/mysql/mysql_ref_functions.asp

La commande UNION : permet de faire l'union de deux ensembles issus d'une commande SELECT.
NB : pour l'utiliser il est primordial que chacune des requetes retourne le meme nombre de colonnes, les memes types et dans le meme ordre (le plus souvent sur la meme table ou sur deux tables qui partagent la meme colonne).

Ex : sélectionner les departements dont la densité n'est pas comprise entre 250 et 7000 en utilsant la commande UNION

```
(
SELECT  *
FROM departement
WHERE densite <=250
)
UNION
(
SELECT  *
FROM departement
WHERE densite >=7000
)
```

Attention : cette commande ne retourne qu'une seule occurrence des enregistrements identiques.

Exp : chercher les citoyens dont le nom de famille est Dubois ou le prenom est stéphane

```
SELECT id, nom, prenom FROM `citoyen` WHERE nom= "Dubois"
UNION
SELECT id, nom, prenom FROM `citoyen` WHERE prenom= "stéphane"
```

Cette requete retourne le resultat suivant :

2	**Dubois**	**stéphane**
6	Dubois	amélie
1	Le grand	stéphane

Alors qu'en exécutant chacune des requetes à part, on obtient, pour la 1ere

2	Dubois	stéphane
6	Dubois	amélie

Et pour la 2eme :

1	Le grand	stéphane
2	Dubois	stéphane

La ligne '2 Dubois stéphane' se repete dans les deux requetes mais elle n'apparait qu'une seule fois dans le resultat final (parce qu'il s'agit de la meme ligne) .

Mais si on utilise UNION ALL, elle va apparaitre 2 fois dans le résultat, on garde meme les doublons.

```
SELECT id, nom, prenom FROM `citoyen` WHERE nom= "Dubois"
UNION ALL
SELECT id, nom, prenom FROM `citoyen` WHERE prenom= "stéphane"
```

2	**Dubois**	**stéphane**
6	Dubois	amélie
1	Le grand	stéphane
2	**Dubois**	**stéphane**

Union all est utile par exemple dans le cas où le meme client achete de plusieurs magasins (ou filiales) et dont chacune detient une liste de clients differentes.

La commande INTERSECT :

La commande intersect existe en SQL mais pas dans MySQL, bien que, il existe une subtilité qui y fait un contournement

```
SELECT id, nom, prenom
FROM citoyen
WHERE
  nom= "Dubois"
    and
  citoyen.id IN (
    SELECT id FROM citoyen WHERE prenom= "stéphane"
  )
```

La commande EXCEPT / MINUS (MINUS dans MySQL) :

Il s'agit de faire deux requetes de selection sur deux ensembles, on prend le résultat du premier sans inclure le résultat du 2eme.

Admettons qu'on a deux tables clients et clients_echeance (clients qui ont dépassé l'échéance). On veut chercher les clients qui n'ont pas dépassé l'échéance.

```
SELECT id, nom, prenom FROM clients
MINUS
SELECT id, nom, prenom FROM clients_echeance
```

Les jointures

Une jointure consiste à utiliser plus qu'une table pour manipuler les données dans la meme requete.

INNER JOIN

Cet type de jointures est le plus utilisé, il consiste à utiliser plus qu'une table et retourne les résultats qui répondent à la clause ON.

la clause ON est généralement utilisée dans le cas de correspondance entre clé primaire et clé étrangère

pour le cas de jointure avec INNER il faut que les valeurs des opérandes de la clause ON soient exactement les memes pour que les enregistrements soient pris en compte

```
SELECT ville.id, ville.nom AS ville, departement.nom AS departement
FROM ville
INNER JOIN departement ON depart_id = departement.id
```

Ex : lister tous les responsables et la responsabilité de chacun.

```
SELECT nom, prenom, role
FROM responsable
INNER JOIN responsabilite ON role_id = responsabilite.id
```

NB : d'autres clauses peuvent être utilisée avec les jointures, **where** par exemple.

Ex : afficher les villes et leurs departements correspondants mais on ne prend que les villes dont la densité dépasse 1000

```
SELECT ville.id, ville.nom AS ville, departement.nom AS departement, ville.densite
FROM ville
INNER JOIN departement ON depart_id = departement.id
WHERE ville.densite > 1000
```

LEFT JOIN

Ce type de jointure permet d'inclure tous les résultats de la table à gauche meme s'il n'y a pas de correspondance avec la clause ON.

```
SELECT responsable.id, nom, prenom, role
FROM responsable
LEFT JOIN responsabilite ON role_id = responsabilite.id
```

NB : vous remarquez dans les résultat que les responsables ayant les id 8 et 9 ne repondent pas à la clause ON mais ils sont dans les resultats car ils figurent dans la table de gauche (1ere table, dans la partie FROM).

RIGHT JOIN (tres rarement utilisée)

Meme chose que LEFT JOIN mais on prend les résultats de la table de droite (2eme table, dans la partie JOIN).

```
SELECT responsable.id, nom, prenom, role
FROM responsable
RIGHT JOIN responsabilite ON role_id = responsabilite.id
```

NB : on affiche les lignes de correspondance de la clause ON et on ajoute aussi les enregistrements de la table de droite (responsable) qui n'ont pas de correspondance.
La ligne président du conseil régional n'a pas d'info correspondantes dans la table responsable, donc les info des colonnes sont à null.

SELF JOIN

Ce type de jointure n'existe pas réellement mais il est tres pratique quand on fait une jointure d'une table avec elle-même

```
SELECT resp.nom, resp.prenom, sup.nom AS superieur
FROM responsable AS resp
LEFT  JOIN responsable AS sup ON resp.sup = sup.id
```

NB : la meme table doit etre renommée dans au moins l'une des clauses FROM ou JOIN, le mieux est de les renommer les deux pour donner à chacune un role explicite et significatif.
Dans cet exemple on a utilisé une left join pour afficher tous les responsables meme ceux qui n'ont pas de supérieur.
Si on a voulu afficher seulement les responsables avec sup, on aurait du mettre inner join.

➔ on utilise une inner ou left join de la table avec elle-même pour simuler une self join qui n'existe pas

CROSS JOIN (tres rarement utilisée)

Ce type de jointure est raremt utilisé, il consiste à faire le produit cartésien des tables utilisées.
Généralement le nombre des lignes est tres elevé, et ca risque de penaliser le serveur.
Exp imaginaire : pour chacune des lignes d'animaux , on veut afficher la liste des aliments.

```
SELECT *
FROM animaux
CROSS JOIN aliments
```

Requetes avancées / sous requetes

NB : quand on veut utiliser une **CONDITION** avec une **FONCTION** SQL, on utilise having à la place de where
En cas de comparaison du resultat d'une fonction dans la clause having, on la compare à une **valeur**.
Pour la comparer à une expression, il faut utiliser where et une sous requete.

Afficher les départements qui appartiennent à la plus grande region

```
SELECT *
FROM departement
WHERE region_id = (
    SELECT id
    FROM region
    WHERE superficie = (
       SELECT MAX(superficie) FROM region
    )
)
```

```
2eme solution
SELECT D.*
FROM departement D
INNER JOIN region R ON D.region_id = R.id
WHERE R.superficie = (
  SELECT MAX(superficie)
  FROM region
);
```

Afficher les regions des departements dont leur densité (du departement) dépasse 5000 habitants, les classer selon la densité par ordre decroissant

```
SELECT departement.id AS id_depart, departement.nom as departement, region.nom AS
region, departement.densite
FROM departement
INNER JOIN region ON region_id = region.id
WHERE departement.densite > 5000
ORDER BY departement.densite DESC
```

Afficher les noms complets des responsables et les noms complets de leurs supérieurs s'ils en ont

```
SELECT responsable.id, concat_ws (' ',responsable.nom,responsable.prenom) AS
responsable, concat_ws (' ',sup.nom,sup.prenom) AS superieur
FROM responsable
INNER JOIN responsable sup ON responsable.sup = sup.id
```

afficher la liste de TOUS les responsables et le nom de son supérieur s'il en a, "aucun" sinon

```
(
  SELECT responsable.id, concat_ws (' ',responsable.nom,responsable.prenom) AS
responsable, concat_ws (' ',sup.nom,sup.prenom) AS superieur
  FROM responsable
  INNER JOIN responsable sup ON responsable.sup = sup.id
)
UNION
(
  SELECT responsable.id, concat_ws (' ',responsable.nom,responsable.prenom) AS
responsable, concat('aucun') AS superieur
  FROM responsable
  WHERE sup IS NULL
)
```

Afficher les noms complets des responsables dont l'age depasse l'age moyen

```
SELECT CONCAT_WS(' ', prenom, nom) AS responsable, (YEAR(CURRENT_DATE)-
YEAR(dat_naiss)) AS age
FROM responsable
WHERE
  ( YEAR(CURDATE())-YEAR(dat_naiss) )
    >
  (   SELECT AVG(YEAR(CURDATE())-YEAR(dat_naiss)) FROM responsable )
```

2eme solution

```
SELECT id, concat_ws(' ', prenom, nom) as responsable, year( curdate() ) -
year(dat_naiss) as age
FROM responsable
HAVING age
      >
      (
        SELECT avg(  year( curdate() ) - year(dat_naiss)  )
        FROM responsable
      )
```

Afficher les departements dont la population de sa région dépasse 6000000 d'habitants

```
SELECT departement.id, departement.nom As departement, region.nom AS region
FROM departement
INNER JOIN region ON region_id = region.id
WHERE region.population > 6000000;
```

Meme ex que le precedent, la différence c'est qu'on a besoin d'avoir l'id, nom du département et la moyenne de la population de sa région

```
SELECT d.id, d.nom, r.nom, avg(r.population) as moyenne_region
FROM departement d
INNER JOIN region r ON region_id =  r.id
GROUP BY id
```

```
HAVING     moyenne_region > 6000000
```

Selectionner les departements dont la population depasse la moyenne, les afficher eux et leurs regions

```
SELECT departement.id, departement.nom, region.nom
FROM departement
INNER JOIN region ON region_id=region.id
WHERE departement.population > (
       SELECT AVG (departement.population) FROM departement
)
```

Insertion de données

```
INSERT INTO nom_table (colonne 1, colonne 2, ..., colonne n)
VALUES (valeur 1, valeur 2, ..., valeur n)
```

Ou on peut faire tout simplement

```
INSERT INTO nom_table  VALUES (valeur 1, valeur 2, ..., valeur n)
```

NB : l'ordre des valeurs doit respecter l'ordre des colonnes de la table.

Ex : insérer un nouveau responsable qui a le role président de la république.

Mise à jour

```
UPDATE nom_table
SET colonne1 = valeur 1, colonne2 = valeur 2, colonne3 = valeur 3
WHERE condition
```

NB : l'opération de maj est potentiellement dangereuse puisqu'elle impacte toutes les lignes si on oublie la condition ou pas les bonnes lignes si on ne met pas la bonne condition.

Ex : modifier la date de naissance du responsable 3 (id = 3), le rendre plus jeune d'une année

```
UPDATE responsable SET dat_naiss = ADDDATE(dat_naiss, INTERVAL 1 YEAR) WHERE id = 3
```

La suppression

```
DELETE FROM nom_table
WHERE condition
```

NB : la suppression est une opération TRES dangereuse puisqu'elle impacte toutes les lignes si on oublie la condition ou pas les bonnes lignes si on ne met pas la bonne condition.

Ex : ajouter un citoyen et le supprimer

```
INSERT INTO citoyen  VALUES (NULL, 'Hidri', 'Ryan', '454654546');
DELETE FROM citoyen WHERE id = 8 ;
```