Remise à niveau SQL

Table des matières

I. Intro

1. Persistance des données
2. CRUD : Create Read Update Delete
3. SQL : Structured Query Language
4. SGBDR : Système de gestion de bases de données relationnelles
   - MySQL / MariaDB
   - SQL Server
   - Oracle
   - PostgreSQL
5. PHPMyAdmin

Site web développé en PHP, offrant une interface graphique pour interagir avec une base de données MySQL/MariaDB.

II. Structure base de données

- Base
  - Table1
    - Champs1 : type
    - Champs2 : type
  - Table2
    - Champs1 : type
      - Champs2 : type
- Requêtes SQL pour la structure

Créer une base de données : CREATE DATABASE nom_de_la_base

- Créer une table :

CREATE TABLE (id INT NOT NULL AUTO_INCREMENT, nom_champ1 VARCHAR(50) NOT NULL, PRIMARY KEY (id))

- Modifier une table (ajouter un champ, modifier un champ, supprimer un champ, ajouter une clé primaire/étrangère :

ALTER TABLE

- Supprimer (une base, une table) :

DROP DATABASE nom_base et DROP TABLE nom_table

- Requêtes pour les données

III. Sélectionner des données : SELECT

- Sélectionner tous les enregistrements dans une table

SELECT * FROM player

SELECT name, email FROM player

- Sélectionner les enregistrements correspondant à une condition

SELECT * FROM player WHERE birthday >= ‘2000-01-01’

- Sélectionner les enregistrements correspondant à plusieurs conditions

SELECT * FROM player WHERE birthday >= ‘2015-01-01’ AND points >= 50

SELECT * FROM player WHERE birthday >= ‘2015-01-01’ AND points >= 50 OR name=’toto’

```
SELECT *
FROM player
WHERE birthday >= ‘2015-01-01’
AND (points >= 50 OR name=’toto’)
```

- Sélectionner les enregistrements dont un champ respecte un « pattern »

% : 0, 1 ou une infinité de caractères

_ : un et un seul caractère

// adresse mail termine par mail.fr

SELECT * FROM player WHERE email LIKE ‘%@mail.fr’

// nom commence par la lettre t

SELECT * FROM player WHERE nom LIKE ‘t%’

// nom commence par la lettre t suivi d’un seul autre caractère

SELECT * FROM player WHERE nom LIKE ‘t_’

- Sélectionner les enregistrements dont la valeur d’un champ se trouve dans une liste prédéfinie

SELECT * FROM player WHERE points = 45 OR points = 65 OR points = 85

SELECT * FROM player WHERE points IN (45, 65, 85)

SELECT * FROM player WHERE name IN (‘toto’, ‘gérard’, ‘abdel’)

- Sélectionner les enregistrements dont la valeur d’un champ se situe entre deux valeurs

SELECT * FROM player WHERE birthday >= ‘2010-01-01’ AND birthday <= ‘2020-12-31’

SELECT * FROM player WHERE birthday BETWEEN ‘2010-01-01’ AND ‘2020-12-31’

IV. ORDER BY : trier les résultats

On peut trier suivant un ou plusieurs champs (clause en fin de requête).

ASC : du plus petit au plus grand

DESC : du plus grand au plus petit

SELECT * FROM player ORDER BY birthday ASC

SELECT * FROM player ORDER BY birthday ASC, name DESC

V. Exercice :

Ecrire les requêtes SQL pour récupérer les joueurs dans la table player :

- les enregistrements dont l'id est supérieur ou égal à 3

SELECT * FROM player WHERE id >= 3

- les enregistrements dont le nom est Dupond et l'adresse mail est dupond@mail.fr

SELECT * FROM player WHERE name = 'Dupond' AND email='dupond@mail.fr'

- les enregistrements triés par date de naissance

SELECT * FROM player ORDER BY birthday ASC

- les enregistrements dont la date de naissance est au moins 2005, triés par nom

SELECT * FROM player WHERE birthday >= '2005-01-01' ORDER BY name

SELECT * FROM player WHERE YEAR(birthday) >= 2005 ORDER BY name

- les enregistrements dont l'adresse email termine soit par sfr.fr ou hotmail.fr

SELECT * FROM player WHERE email LIKE '%@sfr.fr' OR email LIKE '%hotmail.fr'

- les enregistrements dont le nom commence par « s » ou le nombre de points est compris entre 50 et 100
  SELECT * FROM player WHERE name LIKE 's%' OR points >= 50 AND points <= 100
  SELECT * FROM player WHERE name LIKE 's%' OR points BETWEEN 50 AND 100
- Regrouper des enregistrements : GROUP BY

// Sélectionner le nombre de joueurs par points

SELECT points, COUNT(*) FROM player GROUP BY points

// Sélectionner le nombre de points par équipe

```
SELECT SUM(points), team
FROM player
GROUP BY team
```

On peut cumuler plusieurs groupements :

// Sélectionner le nombre de points par équipe par ville

```
SELECT SUM(points), team, zipcode
FROM player
GROUP BY team, zipcode
```

- Conditionner un GROUP BY : HAVING

// Sélectionner le nombre de points par équipe en excluant les équipes qui moins de 60

```
SELECT SUM(points), team
FROM player
GROUP BY team
```

```
HAVING SUM(points)>= 60
```

Là où le WHERE à filtrer les enregistrements, le HAVING sert à filtrer les groupements.

// Sélectionner le nombre de points par équipe en excluant les équipes qui ont au moins 60, et en ne comptant pas le nombre de points des players qui nés après 2019

```
SELECT SUM(points), team
FROM player
WHERE YEAR(birthday) < 2019
GROUP BY team
HAVING SUM(points)>= 60
```

VI. Requête d'insertion : INSERT INTO

```
INSERT INTO player (id, name, email, birthday, points)
VALUES (NULL, 'Bulbizar', 'bubul@mail.fr', NULL, '13');
```

Si un champ peut être null, il n'y a pas obligation à le préciser dans une requête INSERT INTO.

```
INSERT INTO player (id, name, email, points)
VALUES (NULL, 'Bulbizar', 'bubul@mail.fr', '13');
```

On peut ne préciser aucun, dans ce cas, il faut indiquer une valeur pour chaque champ dans VALUES, dans l'ordre des champs de la table.

```
INSERT INTO player
VALUES (NULL, 'Bulbizar', 'bubul@mail.fr', NULL, '13');
```

On peut réaliser plusieurs insertions dans la même requête :

```
INSERT INTO player (id, name, email, birthday, points)
VALUES (NULL, 'Bulbizar', 'bubul@mail.fr', NULL, '13'),
(NULL, 'Pikachu', pika@mail.fr, NULL, '45');
```

VII. Requête de mise à jour : UPDATE

```
UPDATE player SET points = '29' WHERE id = 6;
```

```
UPDATE player SET points = points + 10 WHERE birthday = '2015-07-01';
```

La clause WHERE fonctionne exactement comme dans la requête de sélection. Ici elle sert à préciser quels sont les enregistrements à modifier.

Sans clause WHERE, la modification affectera tous les enregistrements de la table :

```
UPDATE player SET points = '29';
```

Pour modifier plusieurs champs en même temps, il suffit de les séparer par des virgules :

```
UPDATE player SET name = 'toto', email = 'new@mail.fr' WHERE id = 6;
```

VIII. Requête de suppression : DELETE FROM

```
DELETE FROM player WHERE id = 6;
```

```
DELETE FROM player WHERE YEAR(birthday) = 2005;
```

```
DELETE FROM player;
```

IX. Fonctions

1. Fonctions scalaires

Une fonction scalaire, c'est une fonction qui va appliquer un traitement sur un champ de chaque enregistrement.

- UPPER / LOWER : majuscule / minuscule

```
SELECT UPPER(name) FROM player
```

- REVERSE : inverser les caractères
- CONCAT

```
SELECT CONCAT(name, '/', email) FROM player
```

- TRUNCATE : troncature

```
SELECT TRUNCATE(price, 0) FROM product
```

- ROUND : arrondi

```
SELECT ROUND(price, 1) FROM product
```

- YEAR, MONTH, DAY, HOUR, MINUT, SECOND

2. Fonctions d'agrégation

C'est une fonction qui s'applique sur un ensemble de valeur : elles regroupent les lignes.

- COUNT
- SUM
- AVG
- MIN / MAX
- GROUP_CONCAT : lorsqu'un GROUP BY est présent dans la requête, la fonction permet d'obtenir toutes les valeurs qui ont été groupées, séparées par défaut par une virgule.

```
SELECT SUM(points), team, GROUP_CONCAT(name SEPARATOR ',') FROM player GROUP BY team
```

X. Exercices

Ecrire les requêtes qui :

- Sélectionnent les joueurs qui habitent à Paris (75000) et qui ont plus de 75 points, ainsi que les joueurs qui habitent à Lille et qui ont plus de 50 points.

```
SELECT *
FROM player
WHERE zipcode = '75000'
AND points > 75
OR zipcode='59000'
AND points > 50
```

- Sélectionnent le nombre de points par ville, en excluant les villes dont le nombre de points est inférieur à 30.

```
SELECT SUM(points), zipcode
FROM player
GROUP BY zipcode
HAVING SUM(points) >= 30
```

- Sélectionnent les joueurs qui habitent à Arras (62000), qui ont une adresse mail chez wanadoo.fr et dont le nombre de points est compris entre 15 et 25

```
SELECT *
FROM player
WHERE zipcode = '62000'
AND email LIKE '%@wanadoo.fr'
AND points BETWEEN 15 AND 25
```

- Sélectionnent les noms et équipes des joueurs qui habitent à Lille en ayant un nombre de points entre 50 et 60 et qui font partie de l'équipe 1, ainsi que les noms et équipes des joueurs de l'équipe 3 dont le nom commence par « t »

```
SELECT name, team
FROM player
WHERE zipcode = '59000'
AND points BETWEEN 50 AND 60
AND team = 1
OR team = 3
AND name LIKE 't%'
```

- Sélectionnent le nombre de joueur dont le mois de naissance est octobre, en comptant uniquement les joueurs qui ont plus de 30 points

```
SELECT COUNT(id)
FROM player
WHERE MONTH(birthday) = 10
AND points > 30
```

XI. Sous requêtes

Obtenir le joueur qui a le plus grand nombre de points :

En deux requêtes, une pour obtenir le plus grand nombre de points, puis une autre pour récupérer les joueurs qui ont ce plus grand nombre de points

maxPoints = SELECT MAX(points) FROM player ;

SELECT * FROM player WHERE points = maxPoints

En une requête (avec une sous requête) :

```
SELECT * FROM player WHERE points = (SELECT MAX(points) FROM player)
```

XII. Les jointures

- INNER JOIN (jointure interne)

```
SELECT * FROM player INNER JOIN team ON player.team = team.id
```

```
SELECT player.id, player.name, player.points, team.id, team.name
FROM player INNER JOIN team ON player.team = team.id
```

- OUTER JOIN (jointure externe)

```
SELECT * FROM player LEFT OUTER JOIN team ON player.team = team.id
```

```
SELECT * FROM player RIGHT OUTER JOIN team ON player.team = team.id
```

XIII. Alias :

1. Alias de table

Dans une requête on peut créer un alias afin d'éviter d'avoir à écrire le nom de la table à chaque fois qu'on utilise un de ses champs.

Exemple :

```
SELECT P.id, P.name, P.points, T.id, T.name
FROM player P INNER JOIN team T ON P.team = T.id
WHERE T.id = 1
```

2. Alias de colonne

Un alias pour une colonne permet de définir le nom de la colonne dans les résultats de la requête.

```
SELECT P.id AS 'identifiant joueur', P.name, P.points, T.id AS "identifiant équipe", T.name
FROM player P
INNER JOIN team T ON P.team = T.id
WHERE T.id = 1
```

XIV. Création de clé étrangère OneToMany/ManyToOne

Pour informer la base de données que deux tables sont réellement liées, il faut créer une clé étrangère.
Dans PHPMyAdmin, il faut se positionner dans la table qui contient la clé étrangère, « onglet structure », puis cliquez sur le sous-onglet « vue relationnelle ». Il s'agit ensuite de lier les deux colonnes : cela va assurer la cohérence des données.
Dans notre exemple avec les tables « Team » et « Player », cela veut dire qu'on ne pourra plus associer une équipe qui n'existe pas à un joueur.
Attention :

- Les deux colonnes doivent avoir exactement le même type pour pouvoir être liée par une clé étrangère.
- S'il existe déjà des données incohérentes (ex : un joueur lié à une équipe qui n'existe pas), il sera impossible de créer la clé étrangère.

- Exercice
- Créer une nouvelle table « weapon » avec : un id, un nom, une puissance (de 5 à 100), légendaire ou non (booléen)
- Ensuite, vous modifiez la table player pour ajouter une colonne (et clé étrangère) « weapon_id », et vous ajoutez la contrainte de clé étrangère correspondante. Cette colonne peut être nulle.
- Ecrivez la requête qui sélectionne tous les joueurs ainsi que leur arme. Il faut sélectionner aussi les joueurs qui n'ont pas d'arme.

Au lieu de faire une requête pour récupérer tous les joueurs, puis ensuite une requête par joueur pour récupérer son arme :

SELECT * FROM player

SELECT * FROM weapon WHERE id= $player.weapon_id

On peut faire une seule requête avec une jointure :

```
SELECT * FROM player P LEFT OUTER JOIN weapon W ON P.weapon_id = W.id
```

La jointure interne ne correspond à l'énoncé car elle va exclure les joueurs sans arme, la condition de jointure n'étant pas respectée :

```
SELECT * FROM player P INNER JOIN weapon W ON P.weapon_id = W.id
```

XV. Création de clé étrangère ManyToMany

Création de la table Power : id, created_at, name, damage, type

Pour qu'un joueur puisse avoir plusieurs pouvoirs, et qu'un pouvoir puisse être utilisé par plusieurs joueurs, il faut créer une table intermédiaire en base avec deux champs : un champ pour l'identifiant du joueur, et un autre pour l'identifiant du pouvoir.

Ainsi on pourra enregistrer les différentes liaisons.

Les deux champs sont tous deux clé primaire (clé primaire composée), et chacun des champs est également une clé étrangère qui fait référence à un id dans une autre table.

PHPMyAdmin :

- Clé primaire : dans la structure, cochez les deux champs puis cliquez sur le bouton en dessous « Clé primaire » pour créer la clé composée
- Clé étrangère : allez dans l'onglet « Structure » de la table, puis dans le sous-onglet « Vue relationnelle ».

- Jointure pour relation ManyToMany

Jointure interne pour sélectionner tous les joueurs qui ont au moins une pouvoir :

```
SELECT * FROM player
INNER JOIN player_power ON player.id = player_power.player_id
INNER JOIN power ON power.id = player_power.power_id
```

Jointure externe pour sélectionner tous les joueurs même ceux qui n'ont pas de pouvoir :

```
SELECT * FROM player
LEFT OUTER JOIN player_power ON player.id = player_power.player_id
LEFT OUTER JOIN power ON power.id = player_power.power_id
```

Pour sélectionner tous les pouvoirs d'un joueur en particulier (dont vous avez l'id) :

```
SELECT * FROM power
INNER JOIN player_power ON power.id = player_power.power_id
WHERE player_power.player_id = 1
```

XVI. Exercices :

Ecrivez les requêtes suivantes :

- Sélectionner tous les joueurs qui ont le pouvoir 1

```
SELECT * FROM `player`
INNER JOIN player_power ON player.id = player_power.player_id
WHERE player_power.power_id = 1
```

- Sélectionner tous les joueurs qui ont le pouvoir 2 et qui possède une arme (IS NOT NULL)

```
SELECT * FROM `player`
INNER JOIN player_power ON player.id = player_power.player_id
WHERE player_power.power_id = 2
AND player.weapon_id IS NOT NULL
```

- Sélectionner le nombre de points par équipe, en ne comptant pas les joueurs qui n'ont pas d'arme

```
SELECT SUM(points), team
FROM `player`
WHERE weapon_id IS NOT NULL
GROUP BY team
```

- Sélectionner les pouvoirs associés à au moins un joueur de l'équipe 3

```
SELECT * FROM power
INNER JOIN player_power ON power.id = player_power.power_id
INNER JOIN player ON player.id = player_power.player_id
WHERE player.team = 3
```

Version : avec DISTINCT (dédoublonner les résultats du SELECT en double)

```
SELECT DISTINCT power.id, power.name, power.damage
FROM power
INNER JOIN player_power ON power.id = player_power.power_id
INNER JOIN player ON player.id = player_power.player_id
WHERE player.team = 3
```

Version : avec sous-requête

```
SELECT * FROM power
INNER JOIN player_power ON power.id = player_power.power_id
WHERE player_power.player_id IN(SELECT id FROM player WHERE team = 3)
```

XVII. Pour aller plus loin

- TRIGGER : déclencher du SQL lors d'un événement (CREATE/UPDATE/DELETE), juste avant ou après
- PROCEDURE STOCKEES : stocker une fonction SQL personnalisée directement dans le SGBD
- VUES : stocker dans une table « temporaire » les résultats d'une requête
- PRIVILEGES : gérer les droits d'accès au serveur et/ou aux bases et tables