Gestion des Données

On a maintenant notre table, mais on souhaite y insérer des données, créons donc notre premier utilisateur :

Insérer des données

```
INSERT INTO users (username, email, password)
    VALUES ('Maurice', 'mo@gmail.com', 'chaussette');
```

- "**INSERT INTO**" indique que l'on souhaite insérer une donnée dans la table qui suis. "**(...)**" entre parenthèse on indique séparé d'une virgule les colonnes que l'on souhaite ajouter. Ici on a mit des valeurs généré automatiquement pour "**id**", "**active**" et "**createdAt**", on ne s'occupe donc pas de les remplirs et on ajoute uniquement les autres.
- "**VALUES**" indique que ce qui va suivre sont les valeurs à attribuer à notre ligne (**row**)
- "**(...)**" contient les différentes valeurs à donner à nos colonnes, qui doivent être placées dans le même ordre que dans la parenthèse précédente.

Selectionner des données

Si on souhaite voir nos données, on pourra utiliser :

```
SELECT * FROM users;
```

- "**SELECT**" indique que nous souhaitons selectionner des colonnes,
- "*****" indique que l'on veut selectionner toute les colonnes,
- "**FROM**" permet de dire à sql depuis quel table selectionner ces colonnes.

On remarquera que les champs "id", "active" et "createdAt" sont bien rempli.

Mais je n'ai pas toujours besoin d'obtenir toute les informations, par exemple le mot de passe j'en ai besoin lors de la connexion, mais pas si je veux afficher une liste des utilisateurs.

```
SELECT username FROM users;
```

Plutôt que "*****", je peux indiquer le nom des colonnes que je souhaite récupérer.

```
SELECT username, email, createdAt FROM users;
```

Si je souhaite plusieurs colonnes, je peux les séparer d'une virgule.

Ajoutons quelques utilisateurs, copions le contenu de "**addUsers.sql**"; Liens vers le fichier addUsers.sql

On remarquera la possibilité d'ajouté plusieurs lignes à la fois en séparant chaque parenthèse d'une virgule.

Conditionner notre Selection

Si je réutilise une des commandes précédentes, nous verrons que j'obtiens tous les utilisateurs enregistré dans ma bdd.

Quand on en a une dizaine ça va, mais quand on a des milliers, on ne va pas tous les récupérer à chaque fois que je souhaite une information sur un utilisateur.

```
SELECT username FROM users WHERE id = 8;
```

- "**WHERE**" indique que l'on va préciser certaines conditions.
- "**id = 8**" dit ici que je souhaite tout utilisateur dont la colonne id vaut 8.

LIKE et wildcards

```
SELECT username FROM users WHERE email LIKE "%gmail.com";
```

- "**LIKE**" à la différence du "**=**" qui attend une valeur exacte, va chercher une valeur qui contient ce que l'on met ensuite.
- le symbole "**%**" indique "un nombre inconnu de caractère".
- "**%gmail.com**" Ici on recherche toute élément qui se termine par "**gmail.com**".

```
SELECT username FROM users WHERE username LIKE "%il%";
```

Ici c'est donc tout utilisateur dont username contient "**il**";

Mais parfois une seule spécification n'est pas assez et on souhaite en combiner plusieurs.

```
SELECT username FROM users WHERE email LIKE "%gmail.com" AND username LIKE "%il%";
```

- "**AND**" permet de faire plusieurs conditions à la suite. Ici toute personne avec un email en "**gmail.com**" et dont username contient "**il**".

On a donc 3 résultats, testons maintenant avec :

```
SELECT username FROM users WHERE email LIKE "%gmail.com" AND username LIKE
"%_il%";
```

Quel est la différence entre "**%**" et "_", "**%**" signifie n'importe quel caractère entre "**0**" et "**infini**"; "_" signifie n'importe quel caractère, "**1 fois**". Donc ici on demande un nom contenant "**il**" mais ne commençant pas par celui ci.

```
SELECT username FROM users WHERE email LIKE "%gmail.com" OR email LIKE
"%laposte.net";
```

- "**OR**" permet de selectionner selon deux conditions.

```
SELECT username FROM users WHERE NOT email LIKE "%gmail.com";
```

- "**NOT**" permet d'inverser la condition, ici on cherche ceux qui ne possède pas d'adresse gmail.

Autres mots clefs

```
SELECT username FROM users WHERE username <> "Maurice";
```

Dans le même ordre d'idée "**<>**" ou "**!=**" selon les versions de "**SQL**", permet de selectionner quelque chose différent de la valeur indiqué.

Pour selectionner une tranche d'utilisateur, on peut faire comme ceci :

```
SELECT username FROM users WHERE id >= 5 AND id <= 8;
```

Ou bien utiliser :

```
SELECT username FROM users WHERE id BETWEEN 5 AND 8;
```

- "**BETWEEN**" indique que l'on cherches des entrées qui se trouvent entre deux valeur.

```
SELECT username FROM users WHERE username IN ("Maurice", "Hypolite", "Florestan");
```

- "**IN**" permet de rechercher la correspondance de la colonne dans une liste de valeur donnée. Notez que l'on peut très bien faire une autre requête sql dans ce IN, par exemple :

```
SELECT username FROM users WHERE username IN (
    SELECT username FROM users WHERE email LIKE "%gmail%"
    );
```

Cet exemple n'a aucun intérêt mais cela peut être utile dans une BDD plus massive.

```
SELECT username FROM users WHERE password IS NULL;
```

- "**IS NULL**" ou "**IS NOT NULL**" permet de vérifier si un champ est NULL ou non.

Trier nos Données

On souhaite parfois recevoir nos résultats ordonné d'une certaine façon :

```
SELECT id, username, createdAt FROM users ORDER BY createdAt;
```

"**ORDER BY**" permet de trier nos résultats selon certains critères Il se place après le "**WHERE**".

On reçois nos utilisateurs triés par date de création ascendante. Mais on peut être plus précis:

```
SELECT * FROM users ORDER BY active DESC, username ASC;
```

Ici on va récupérer nos utilisateurs par active descandant "**DESC**" et si des valeurs sont égales, elles seront triées par username "**ASC**".

(par defaut une valeur est trié par "**ASC**" mais on peut l'écrire pour le préciser.)

LIMIT et OFFSET

On peut limiter le nombre de résultat obtenu avec "**LIMIT**":

```
SELECT * FROM users LIMIT 5;
```

"**OFFSET**" permet d'ignorer les premiers résultats, (ici on ignore les trois premiers.)

```
SELECT * FROM users LIMIT 5 OFFSET 3;
```

Ces deux propriétés sont souvent utiliser pour la pagination, par exemple on veut 10 résultats par page `LIMIT 10` puis :

- Page 1 : `OFFSET 0`
- Page 2 : `OFFSET 10`
- Page 3 : `OFFSET 20`

Supprimer des Données

J'aimerais bien supprimer mon utilisateur "Maurice" dans ce cas :

```
DELETE FROM users;
```

STOP !!!! Si vous faites ça, c'est toute votre table qui est supprimé !

```
DELETE FROM users WHERE username = "Maurice";
```

Ici on supprimera bien que l'utilisateur "*Maurice*", faites bien attention de ne pas oublier le "**WHERE**".

Mettre à jour des Données

Il me faudrait mettre à jour mes utilisateurs.

```
UPDATE users SET active = 1, createdAt = CURRENT_TIMESTAMP;
```

STOP !!!! encore une fois, ici vous allez mettre tous vos utilisateurs à jour.

```
UPDATE users SET active = 1, createdAt = CURRENT_TIMESTAMP WHERE email LIKE
"%gmail.com";
```

Pour l'update aussi on oublie surtout le pas "**WHERE**";

Différents type de requêtes

Distinct

J'aimerais connaître les différent mots de passes utilisé par mes utilisateurs (à ne pas faire en vrai, de toute façon ils seront hashés).

```
SELECT password FROM users;
```

Certe j'aurais tous les mots de passes mais aussi des doublons. Pour éviter cela :

```
SELECT DISTINCT password FROM users;
```

"**DISTINCT**" permettra de récupérer uniquement les valeurs différentes de notre colonne.

Utile par exemple si on veut connaître les différents pays d'où viennent nos utilisateurs.

FONCTIONS

Quelques fonctions pratique sont disponible, telle que :

```
SELECT COUNT(id) FROM users;
```

"**COUNT()**" nous permet de compter le nombre de résultat.

```
SELECT AVG(id) FROM users;
```

"**AVG()**" permet de faire une moyenne d'une colonne.

```
SELECT SUM(id) FROM users;
```

"**SUM()**" permet de faire la somme d'une colonne.

```
SELECT MIN(id) FROM users;
```

"**MIN()**" retourne la plus petite valeur de la colonne.

```
SELECT MAX(id) FROM users;
```

"**MAX()**" retourne la plus grande valeur de la colonne.

```
SELECT MONTH(createdAt) FROM `users`;
```

"**MONTH()**" retourne uniquement le mois d'une date. (il existe la même choses pour l'année, les minutes...)

```
SELECT CONCAT(username, " : ", email) FROM users;
```

"**CONCAT()**" permet de concatener les résultats d'une requête en une seule colonne.

```
SELECT CAST(
    (SELECT COUNT(id) FROM users WHERE email LIKE "%gmail%")
    /
    (SELECT COUNT(id) FROM users)
    *100 AS INT
    );
```

"**CAST()**" permet de changer la valeur d'un champs. On indique la valeur à changer puis "**AS**" et enfin le type (ICI on a un **CAST(calcul AS INT)**)

Sans "**CAST**" le calcul pourrait être un "**FLOAT**", un nombre à virgule. Avec, on obtient un "**INT**", un nombre sans virgule.

ALIAS

Si on fait attention, on voit que nos résultats précédents sont nommé "*COUNT(id), AVG(id),...*" ce n'est pas très lisible. (encore plus pour le *CAST()*) De plus quand on utilisera un langage back, il nous faudra utiliser ces noms pour selectionner les données.

Mais il est possible de renommer les colonnes et les tables pour qu'elles apparaissent différement dans nos résultats.

```
SELECT COUNT(id) AS TotalUser FROM users;
```

Grâce à "**AS**" notre résultat se nomme "**TotalUser**".

Vous pouvez essayer sur le "**CAST()**" de tout à l'heure, le résultat sera bien plus lisible.

On peut aussi "**Alias**" une table sans le mot "**AS**":

```
SELECT username FROM users u;
```

Comme cela ça peut sembler inutile, mais on verra l'utilité avec les "**jointures**".

GROUP BY

Je souhaite compter quels utilisateurs utilise "*gmail*" et combien ne l'utilise pas, pour cela je peux faire :

```sql
SELECT id, (email LIKE '%gmail%') AS gmail FROM `users`;
```

Je vais avoir les id de chaque utilisateur et une colonne gmail indiquant un **boolean**. Mais si je compte les id :

```sql
SELECT COUNT(id), (email LIKE '%gmail%') AS gmail FROM `users`;
```

Le résultat ne sera pas celui voulu car il comptera le total, et si j'ajoute une close "**WHERE**", je n'aurais que ceux qui ont gmail.

Il est possible de grouper certains résultats avec "**GROUP BY**" :

```sql
SELECT COUNT(id), (email LIKE '%gmail%') AS gmail
    FROM `users`
    GROUP BY gmail;
```

On aura alors les résultats de la colonne "**gmail**" groupé et le total d'id pour chacun affiché. Notons qu'il est possible de faire plusieurs groupes si séparé d'une virgule.

HAVING

Il est possible de n'afficher que certains résultat de groupe, comme avec le mot clef "**WHERE**" mais spécifiquement pour les groupes, et cela avec "**HAVING**" :

```sql
SELECT COUNT(id), (email LIKE '%gmail%') AS gmail
    FROM `users`
    GROUP BY gmail
    HAVING COUNT(id) >= 5;
```

Ici on obtiens seulement les lignes où le "**COUNT(id)**" est plus grand ou égale à 5.