PARTIE II

Bases de données (SQL)

MySQL

M. AATILA

2021/2022

Les interrogations en SQL (MySQL)

L'utilisation la plus courante de SQL consiste à lire des données issues de la base de données. Cela s'effectue grâce à la commande SELECT, qui retourne des enregistrements dans un tableau de résultat. Cette commande peut sélectionner une ou plusieurs colonnes d'une table.

L'ordre SELECT possède six clauses différentes, dont seules les deux premières sont obligatoires. Elles sont données ci-dessous, dans l'ordre dans lequel elles doivent apparaître, quand elles sont utilisées :

- **SELECT ...**
- **FROM ..**
- **WHERE ...**
- **GROUP BY ...**
- **HAVING ...**
- **ORDER BY ...**

1. Clause SELECT :

Cette clause permet d'indiquer quelles colonnes, ou quelles expressions doivent être retournées par l'interrogation.

a. Commande de base :

L'utilisation basique de cette commande s'effectue de la manière suivante :

```
SELECT nom_du_champ
FROM nom_de_table ;
```

Cette requête va sélectionner (SELECT) le champ « nom_du_champ » provenant (FROM) du tableau appelé « nom_de_table ».

b. Exemple :

Imaginons une base de données comportant une table appelée « client » qui contient des informations sur les clients d'une entreprise.

- **Table « client » :**

identifiant	prenom	nom	ville
1	Pierre	Dupond	Paris
2	Sabrina	Durand	Nantes
3	Julien	Martin	Lyon
4	David	Bernard	Marseille
5	Marie	Leroy	Grenoble

Si l'on veut avoir la liste de toutes les villes des clients, il suffit d'effectuer la requête suivante :

```
SELECT ville
FROM client ;
```

- **Résultat :**

ville
Paris
Nantes
Lyon
Marseille
Grenoble

- **Obtenir plusieurs colonnes :**

Avec la même table client il est possible de d'afficher plusieurs colonnes à la fois. Il suffit tout simplement de séparer les noms des champs souhaités par une virgule. Pour obtenir les prénoms et les noms des clients il faut alors faire la requête suivante :

```
SELECT prenom, nom
FROM client ;
```

- **Résultat :**

prenom	nom
Pierre	Dupond
Sabrina	Durand
Julien	Martin
David	Bernard
Marie	Leroy

- **Obtenir toutes les colonnes d'une table :**

Il est possible de retourner automatiquement toutes les colonnes d'un tableau sans avoir à connaître le nom de toutes les colonnes. Au lieu de lister toutes les colonnes, il faut simplement utiliser le caractère « * » (**étoile**). C'est un joker qui permet de sélectionner toutes les colonnes. Il s'utilise de la manière suivante :

```
SELECT * FROM client ;
```

- **Résultat :** Cette requête retourne exactement les mêmes colonnes qu'il y a dans la base de données. Dans notre cas, le résultat sera donc :

identifiant	prenom	nom	ville
1	Pierre	Dupond	Paris
2	Sabrina	Durand	Nantes
3	Julien	Martin	Lyon
4	David	Bernard	Marseille
5	Marie	Leroy	Grenoble

c. Alias et select distinct :

L'utilisation de la commande SELECT en SQL permet de lire toutes les données d'une ou plusieurs colonnes. Cette commande peut potentiellement afficher des lignes en doubles. Pour éviter des redondances dans les résultats il faut simplement ajouter **DISTINCT** après le mot SELECT.

Lors de l'affichage du résultat de la requête on peut changer les titres des colonnes, uniquement lors de l'affichage sur écran et non pas au niveau de la table, en utilisant des **ALIAS** exprimés par le mot clé **AS**.

- **Exemple :**

Prenons le cas concret d'une table « client » qui contient des noms et prénoms :

identifiant	prenom	nom
1	Pierre	Dupond
2	Sabrina	Bernard
3	David	Durand
4	Pierre	Leroy
5	Marie	Leroy

En utilisant seulement **SELECT (Requête 1)** tous les noms sont retournés, or la table contient plusieurs fois le même prénom (Pierre). Pour sélectionner uniquement les prénoms uniques il faut utiliser une requête avec un **DISTINCT (Requête 2**) :

<table>
<tr><th>REQUETE 1</th><th>REQUETE 2</th></tr>
<tr><td>SELECT Prenom from client ;</td><td>SELECT DISTINCT Prenom AS "Prénoms des clients" from Client ;</td></tr>
<tr><td>• RESULTAT 1 :
<table><tr><th>Prenom</th></tr><tr><td>Pierre</td></tr><tr><td>Sabrina</td></tr><tr><td>David</td></tr><tr><td>Pierre</td></tr><tr><td>Marie</td></tr></table></td><td>• RESULTAT 2 :
<table><tr><th>Prénoms des clients</th></tr><tr><td>Pierre</td></tr><tr><td>Sabrina</td></tr><tr><td>David</td></tr><tr><td>Marie</td></tr></table></td></tr>
</table>

2. Clause FROM :

La clause **FROM** permet de mentionner la liste des tables participant à l'interrogation.

- **Syntaxe :**

FROM table1 [synonyme1] , table2 [synonyme2] , ...

- **Exemple :**

Liste des noms des clients de l'entreprise, ici on doit utiliser la table Client dans la clause FROM.

```
SELECT * FROM Client;
```

➔ Il est possible de lancer des interrogations utilisant plusieurs tables à la fois.

NB : Les deux clauses SELECT et FROM sont indispensables dans n'importe quelle requête !

3. Clause WHERE :

La clause **WHERE** permet de spécifier quelles sont les lignes à sélectionner dans une table ou dans le produit cartésien de plusieurs tables.

- **Syntaxe :**

La commande **WHERE** s'utilise en complément à une requête utilisant SELECT. La façon la plus simple de l'utiliser est la suivante :

```
SELECT nom_colonnes
FROM nom_table
WHERE condition ;
```

- **Exemple :**

Imaginons une table appelée « client » qui contient le nom des clients, le nombre de commandes qu'ils ont effectués et leur ville :

id	nom	nbr_commande	ville
1	Paul	3	paris
2	Maurice	0	rennes
3	Joséphine	1	toulouse
4	Gérard	7	paris

Pour obtenir seulement la liste des clients qui habitent à Paris, il faut effectuer la requête suivante :

```
SELECT *
FROM client
WHERE  ville = 'paris' ;
```

- **Résultat :**

id	nom	nbr_commande	ville
1	Paul	3	paris
4	Gérard	7	paris

Attention : dans notre cas tout est en minuscule donc il n'y a pas eu de problème. Cependant, si une table contient des villes écrites en Majuscules, dans la condition il faut prendre ceci en considération. IL faut faire attention aux majuscules et minuscules pour les données des tables.

- **Opérateurs de comparaisons :**

Il existe plusieurs opérateurs de comparaisons. La liste ci-jointe présente quelques-uns des opérateurs les plus couramment utilisés.

Opérat	Description	Opérat	Description
=	Égale	<=	Inférieur ou égale à
<>	Pas égale	IN	Liste de plusieurs valeurs possibles
!=	Pas égale	BETWEEN	Valeur comprise dans un intervalle donnée

>	Supérieur à	LIKE	Recherche en spécifiant le début, milieu ou fin d'un mot
<	Inférieur à	IS NULL	Valeur est nulle
>=	Supérieur ou égale à	IS NOT NULL	Valeur n'est pas nulle

a. Opérateurs AND & OR :

Une requête SQL peut être restreinte à l'aide de la condition WHERE. Les opérateurs logiques **AND** et **OR** peuvent être utilisés au sein de la commande WHERE pour combiner des conditions.

- **Syntaxe d'utilisation des opérateurs AND et OR :**

Les opérateurs sont à ajoutés dans la condition WHERE. Ils peuvent être combinés à l'infini pour filtrer les données comme souhaités.

L'opérateur **AND** permet de s'assurer que la condition1 **ET** la condition2 sont vraies et L'opérateur **OR** vérifie quant à lui que la condition1 **OU** la condition2 est vraie :

- **Exemple :**

1. Pour obtenir seulement la liste des clients qui habitent à Paris et qui ont effectué plus que 4 commandes, il faut effectuer la requête suivante :

```
SELECT * FROM Client
WHERE ville='paris' AND nbr_commande>=4 ;
```

- **Résultat :**

id	nom	nbr_commande	ville
4	Gérard	7	paris

2. Pour obtenir seulement la liste des clients qui habitent à Paris ou qui n'ont pas fait de commandes, il faut effectuer la requête suivante :

```
SELECT * FROM Client
WHERE ville='paris' OR nbr_commande = 0 ;
```

- **Résultat :**

id	nom	nbr_commande	ville
1	Paul	3	paris
2	Maurice	0	rennes
4	Gérard	7	paris

Remarque : On peut combiner dans une requête des AND et des OR.

b. Opérateur IN :

L'opérateur logique **IN** dans SQL s'utilise avec la commande WHERE pour vérifier si une colonne est égale à une des valeurs comprises dans un ensemble de valeurs déterminées. C'est une méthode simple pour vérifier si une colonne est égale à une valeur **OU** une autre valeur **OU** une autre valeur et ainsi de suite, sans avoir à utiliser de multiple fois l'opérateur **OR**.

- **Exemple :**

Pour obtenir seulement la liste des clients qui habitent à Paris ou à rennes, il faut effectuer l'une des requêtes suivantes :

```
SELECT * FROM Client
```

```
WHERE ville='paris' OR  ville='rennes' ;
```

Ou bien

```
SELECT * FROM Client
WHERE ville IN ('paris' , 'rennes')  ;
```

- **Résultat :**

Les 2 requêtes retournent les mêmes résultats, l'une utilise l'opérateur IN, tandis que l'autre utilise plusieurs OR.

id	nom	nbr_commande	ville
1	Paul	3	paris
2	Maurice	0	rennes
4	Gérard	7	paris

c. Opérateur BETWEEN :

L'opérateur **BETWEEN** est utilisé dans une requête SQL pour sélectionner un intervalle de données dans une requête utilisant WHERE. L'intervalle peut être constitué de chaînes de caractères, de nombres ou de dates.

- **Exemple :**

Pour obtenir seulement la liste des clients qui ont effectué entre 3 et 9 commandes, on peut effectuer la requête suivante :

```
SELECT * FROM Client
WHERE nbr_commande BETWEEN 3 AND 9 ;
```

- **Résultat :**

id	nom	nbr_commande	ville
1	Paul	3	paris
4	Gérard	7	paris

d. Opérateur LIKE :

L'opérateur **LIKE** est utilisé dans la clause WHERE des requêtes SQL. Il est par exemple possible de rechercher les enregistrements dont la valeur d'une colonne commence par telle ou telle lettre. Les modèles de recherches sont multiples.

Pour l'opérateur LIKE on dispose de 2 caractères jokers qui sont :

- ✓ **%** : remplaçant une chaîne de caractères de 0 ou plusieurs caractères.
- ✓ **_** : le caractère « _ » (underscore) remplaçant un seul caractère.

- **Exemples :**

Pour afficher la liste des clients dont le nom se termine par un 'e', il faut effectuer la requête suivante :

```
SELECT * FROM Client
WHERE nom LIKE '%e' ;
```

- **Résultat :**

id	nom	nbr_commande	ville
2	Maurice	0	rennes
3	Joséphine	1	toulouse

Pour afficher la liste des clients dont le premier caractère du nom est un 'M' et le troisième caractère du nom toujours est un 'u', i faut effectuer la requête suivante :

```
SELECT * FROM Client
WHERE nom LIKE 'M_ u%' ;
```

- **Résultat :**

id	nom	nbr_commande	ville
2	Maurice	0	rennes

e. Opérateurs IS NULL / IS NOT NULL :

Dans le langage SQL, l'opérateur **IS** permet de filtrer les résultats qui contiennent la valeur **NULL**. Cet opérateur est indispensable car la valeur NULL est une valeur inconnue et ne peut par conséquent pas être filtrée par les opérateurs de comparaison (égal, inférieur, supérieur ou différent).

- **Exemple :**

Pour afficher la liste des clients ayant la valeur NULL pour le nombre de commandes, il faut effectuer la requête suivante :

```
SELECT * FROM Client
WHERE nbr_commande IS NULL ;
```

➔ Pour notre exemple, cette requête ne retournera pas de résultat puisqu'aucun client n'a la valeur NULL pour le nombre de commande.

f. Les jointures naturelles :

La clause WHERE peut aussi être utilisée pour faire des jointures et des sous-interrogations (une des valeurs utilisées dans un WHERE provient d'une requête SELECT emboîtée). Quand on précise plusieurs tables dans la clause FROM, on obtient le produit cartésien des tables.

La jointure naturelle permet de rassembler des informations provenant de plusieurs tables. Chaque ligne d'une table figurant dans cette jointure doit avoir un correspondant dans les autres tables.

Soit la base de données, contenant les informations concernant les commandes client pour une entreprise donnée, suivante. Supposant que cette base de données ne contient que 2 tables Client et Commande comme dans le tableau suivant :

Table Client

IdCl	Nom	Ville
1	Paul	paris
2	Maurice	rennes
3	Joséphine	toulouse
4	Gérard	paris

Table Commande

Idcmde	DatCmde	IdCl
1	01/05/2005	2
2	23/12/2011	1
3	11/09/2013	3
4	30/12/2016	1

- **Exemple :**

Pour afficher la liste des clients ayant effectué une commande le mois de Mai 2005, il faut effectuer la requête suivante :

```
SELECT IdCL, Nom, Ville, DateCmde
FROM Client Cl, Commande Cmde
WHERE (Cl.CodeCL = Cmde.CodeCL)  /*Jointure utilisant les synonymes des
                                   tables Client et Commande */
And (commande.DateC between '01/05/2005' and '31/05/2005');
```

- **Résultat :**

IdCl	Nom	Ville	DatCmde
2	Maurice	rennes	01/05/2005

g. Les requêtes imbriquées :

Une caractéristique puissante de SQL est la possibilité qu'un prédicat employé dans une clause WHERE comporte un SELECT emboîté. Par exemple, la sélection des clients ayant même ville que Paul peut s'écrire en faisant une requête imbriquée sur la table Client :

```
SELECT *  FROM Client
WHERE Ville = (SELECT Ville  FROM Client WHERE NomCL = 'Paul');
```

4. Clause GROUP BY :

La commande **GROUP BY** est utilisée en SQL pour grouper plusieurs résultats et utiliser une fonction de totaux sur un groupe de résultat. Sur une table qui contient toutes les ventes d'un magasin, il est par exemple possible de regrouper les ventes par clients identiques et d'obtenir le coût total des achats pour chaque client.

- **Syntaxe d'utilisation de GROUP BY**

De façon générale, la commande **GROUP BY** s'utilise de la façon suivante :

```
SELECT colonne1, FonctionDeGroupe(colonne2)
FROM table
GROUP BY colonne1 ;
```

a. Fonctions de groupe :

Il existe plusieurs fonctions qui peuvent être utilisées pour manipuler plusieurs enregistrements, il s'agit des fonctions d'agrégations statistiques, les principales sont les suivantes :

AVG	: moyenne
SUM	: somme
MIN	: plus petite des valeurs
MAX	: plus grande des valeurs
COUNT(*)	: nombre de lignes

- **Exemple d'utilisation :**

Prenons en considération une table « achat » qui résume les ventes d'une boutique :

idAchat	Client	Tarif	Date
1	Pierre	102	2012-10-23
2	Simon	47	2012-10-27
3	Marie	18	2012-11-05
4	Marie	20	2012-11-14
5	Pierre	160	2012-12-03

Pour obtenir le coût total de chaque client en regroupant les commandes des mêmes clients, il faut utiliser la requête suivante :

```
SELECT client, SUM(tarif)
FROM achat
GROUP BY client ;
```

- **Résultat :**

Client	SUM(tarif)
Pierre	262
Simon	47
Marie	38

NB : la seule colonne qu'on peut mettre au niveau du SELECT dans ce cas c'est la colonne utilisée dans la clause GROUP BY.

5. Clause 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**().

- **Syntaxe :**
  L'utilisation de **HAVING** s'utilise de la manière suivante :

```
SELECT colonne1, SUM(colonne2)
FROM nom_table
GROUP BY colonne1
HAVING fonction(colonne2) operateur valeur ;
```

- **Exemple :** Si dans la table achat on souhaite récupérer la liste des clients qui ont commandé plus de 40 €, toute commandes confondues alors il est possible d'utiliser la requête suivante :

```
SELECT client, SUM(tarif)
FROM achat
GROUP BY client
HAVING SUM(tarif) > 40 ;
```

- **Résultat :**

Client	SUM(tarif)
Pierre	262
Simon	47

6. Clause ORDER BY :

La commande **ORDER BY** permet de trier les lignes dans un résultat d'une requête SQL. Il est possible de trier les données sur une ou plusieurs colonnes, par ordre ascendant ou descendant.

- **Syntaxe :**

Une requête où l'on souhaite filtrer l'ordre des résultats utilise la commande **ORDER BY** de la sorte suivante :

```
SELECT colonne1, colonne2
FROM table
ORDER BY colonne1 ;
```

Par défaut les résultats sont classés par ordre ascendant, toutefois il est possible d'inverser l'ordre en utilisant le suffixe **DESC** après le nom de la colonne. Par ailleurs, il est possible de trier sur plusieurs colonnes en les séparant par une virgule. Une requête plus élaboré ressemblerais alors cela :

```
SELECT colonne1, colonne2, colonne3
FROM table
ORDER BY colonne1 DESC, colonne2 ASC ;
```

- **Exemple :**

Pour récupérer la liste de ces achats par ordre décroissant du IdAchat, il est possible d'utiliser la requête suivante :

```
SELECT *
FROM Achat
ORDER BY IdAchat DESC ;
```

- **Résultat :**

idAchat	Client	Tarif	Date
5	Pierre	160	2012-12-03
4	Marie	20	2012-11-14
3	Marie	18	2012-11-05
2	Simon	47	2012-10-27
1	Pierre	102	2012-10-23