Chapitre 5
Techniques avancées avec MySQL

1. Grouper les données

Parfois, vous pouvez avoir besoin de calculer une valeur pour un niveau de regroupement :

– chiffre d'affaires cumulé par région,

– salaire moyen par département.

Pour ce genre d'interrogation, vous allez pouvoir utiliser des fonctions d'agrégat (`SUM`, `AVG`, etc.) et regrouper les données grâce à la clause `GROUP BY` ; en complément, le résultat final, après regroupement, peut être restreint avec la clause `HAVING`.

Syntaxe

```
SELECT expression[,...] | *
FROM nom_table
[WHERE conditions]
GROUP BY expression [,...]
[HAVING conditions]
[ORDER BY expression [ASC | DESC][,...]]
[LIMIT [offset,] nombre_lignes]
```

La clause `GROUP BY` s'intercale entre les clauses `WHERE` et `ORDER BY` (si elles sont présentes). Elle spécifie les expressions utilisées pour effectuer le regroupement.

`expression` peut être :

– une colonne,
– une expression basée sur des colonnes,
– un alias de colonne,
– un numéro correspondant à la position d'une expression de la clause `SELECT` (syntaxe déconseillée et obsolète).

Dans les anciennes versions, le résultat de la requête était trié par défaut en ordre croissant sur les différentes expressions de la clause `GROUP BY` ; l'ordre du tri de chaque niveau de regroupement pouvait être défini explicitement avec des options `ASC` et `DESC`. Depuis la version 5.7, se baser sur le tri implicite provoqué par la clause `GROUP BY` était déprécié, et depuis la version 8.0.13, les options `ASC` et `DESC` de la clause `GROUP BY` ont été supprimées. Pour être certain d'avoir un ordre de tri bien précis, il est recommandé d'utiliser une clause `ORDER BY` explicite.

La plupart du temps, vous mettrez dans la clause `GROUP BY` toutes les expressions de la clause `SELECT` qui ne comportent pas de fonction d'agrégat. C'est le fonctionnement habituel pour respecter le SQL standard.

Exemples

```
mysql> -- Nombre de livres par collection.
mysql> SELECT id_collection,COUNT(*)
    -> FROM livre
    -> GROUP BY id_collection;
+---------------+----------+
| id_collection | COUNT(*) |
+---------------+----------+
|             1 |        7 |
|             3 |        1 |
|             5 |        1 |
|             4 |        1 |
+---------------+----------+
4 rows in set (0.00 sec)

mysql> -- Nombre de livres et nombre moyen de pages par collection
mysql> -- (avec le nom de la collection au lieu de l'identifiant).
mysql> SELECT
    ->    col.nom collection,
    ->    COUNT(liv.id) nb_livres,
    ->    ROUND(AVG(liv.nombre_pages)) nb_pages_moy
    -> FROM
    ->    livre liv JOIN collection col
    ->    ON (liv.id_collection = col.id)
    -> GROUP BY
```

```
    ->    col.nom
    -> ORDER BY
    ->    nb_livres; -- tri sur le nombre de livres
+--------------------------+-----------+--------------+
| collection               | nb_livres | nb_pages_moy |
+--------------------------+-----------+--------------+
| Les TP Informatiques     |         1 |          302 |
| Epsilon                  |         1 |          552 |
| Coffret Technique        |         1 |         1166 |
| Ressources Informatiques |         7 |          579 |
+--------------------------+-----------+--------------+
4 rows in set (0.00 sec)

mysql> -- Nombre de livres par tranche d'années et collection.
mysql> SELECT
    ->   CASE
    ->     WHEN annee_parution BETWEEN 2005 AND 2011
    ->     THEN '2005-2011'
    ->     WHEN annee_parution BETWEEN 2012 AND 2019
    ->     THEN '2012-2019'
    ->   END tranche_annee,
    ->   col.nom collection,
    ->   COUNT(liv.id) nb_livres
    -> FROM
    ->   livre liv JOIN collection col
    ->   ON (liv.id_collection = col.id)
    -> GROUP BY -- utilisation des alias de colonne
    ->   tranche_annee,
    ->   collection;
    -> ORDER BY
    ->   tranche_annee,
+---------------+--------------------------+-----------+
| tranche_annee | collection               | nb_livres |
+---------------+--------------------------+-----------+
| 2005-2011     | Les TP Informatiques     |         1 |
| 2005-2011     | Ressources Informatiques |         2 |
| 2012-2019     | Coffret Technique        |         1 |
| 2012-2019     | Epsilon                  |         1 |
| 2012-2019     | Ressources Informatiques |         5 |
+---------------+--------------------------+-----------+
5 rows in set (0.00 sec)
```

Avec MySQL, vous pouvez avoir dans la clause `SELECT` des expressions qui n'utilisent pas de fonction d'agrégat et qui ne sont pas reprises dans la clause `GROUP BY`.

Exemple

```
mysql> SELECT
    ->   col.nom collection, -- non présent dans le GROUP BY
    ->   col.prix_ht, -- non présent dans le GROUP BY
    ->   COUNT(liv.id) nb_livres
    -> FROM
    ->   livre liv JOIN collection col
    ->   ON (liv.id_collection = col.id)
    -> GROUP BY
    ->   col.id;
+--------------------------+---------+-----------+
| collection               | prix_ht | nb_livres |
+--------------------------+---------+-----------+
| Ressources Informatiques |   28.48 |         7 |
| Les TP Informatiques     |   25.71 |         1 |
| Epsilon                  |   51.90 |         1 |
| Coffret Technique        |   54.19 |         1 |
+--------------------------+---------+-----------+
4 rows in set (0.00 sec)
```

Cette possibilité de MySQL est intéressante et permet de simplifier l'écriture de la requête et d'améliorer les performances lorsque l'expression concernée est unique pour chaque groupe (c'est le cas sur l'exemple précédent où le nom et le prix de la collection sont uniques pour chaque identifiant de collection). Par contre, il est déconseillé d'utiliser cette possibilité si l'expression concernée n'est pas unique dans le groupe ; MySQL va retourner n'importe quelle valeur pour le groupe et le résultat obtenu n'a en général pas vraiment de sens. Si le mode SQL `ONLY_FULL_GROUP_BY` est actif (c'est le cas par défaut depuis la version 5.7.5), MySQL génère une erreur si les clauses `SELECT`, `HAVING` ou `ORDER BY` référencent des colonnes qui n'utilisent pas de fonction d'agrégat ou qui ne sont pas présentes dans la clause `GROUP BY`, et qui ne sont pas fonctionnellement dépendantes des colonnes présentes dans la clause `GROUP BY`.

Exemple

```
mysql> SELECT
    ->   col.nom collection,
    ->   liv.annee_parution, -- pas unique dans le groupe !
    ->   COUNT(liv.id) nb_livres
    -> FROM
    ->   livre liv JOIN collection col
    ->   ON (liv.id_collection = col.id)
    -> GROUP BY
    ->   col.nom;
ERROR 1055 (42000): Expression #2 of SELECT list is not in GROUP BY clause and
contains nonaggregated column 'eni.liv.annee_parution' which is not functionally
dependent on columns in GROUP BY clause; this is incompatible with
sql_mode=only_full_group_by
```

Dans ce genre de situation, il peut être intéressant d'utiliser les fonctions `MIN` ou `MAX` pour obtenir une valeur spécifique « contrôlée » de l'expression (« le plus grand », « le plus petit », « le premier », « le dernier », etc.).

Exemple

```
mysql> SELECT
    ->   col.nom collection,
    ->   MIN(liv.annee_parution) premiere_parution,
    ->   COUNT(liv.id) nb_livres
    -> FROM
    ->   livre liv JOIN collection col
    ->   ON (liv.id_collection = col.id)
    -> GROUP BY
    ->   col.nom
    -> ORDER BY
    ->   col.nom;
+-------------------------+-------------------+-----------+
| collection              | premiere_parution | nb_livres |
+-------------------------+-------------------+-----------+
| Coffret Technique       |              2019 |         1 |
| Epsilon                 |              2016 |         1 |
| Les TP Informatiques    |              2007 |         1 |
| Ressources Informatiques |              2008 |         7 |
+-------------------------+-------------------+-----------+
4 rows in set (0.00 sec)
```

Pour restreindre le résultat final, avec des conditions utilisant les fonctions d'agrégat, vous devez utiliser la clause `HAVING`. En effet, une clause `WHERE` ne peut pas comporter de condition qui utilise une fonction d'agrégat : c'est trop "tôt" dans le traitement de la requête ; les groupes n'ont pas encore été constitués.

Les conditions d'une clause `HAVING` sont semblables à celles d'une clause `WHERE` avec les particularités suivantes :

– elles peuvent utiliser des fonctions d'agrégat ;

– elles peuvent utiliser les alias définis dans la clause `SELECT`.

Exemple

```
mysql> -- Afficher uniquement les lignes
mysql> -- qui ont au moins deux livres.
mysql> SELECT
    ->   (nombre_pages DIV 100) * 100 tranche_pages,
    ->   COUNT(*) nb_livres,
    ->   MAX(annee_parution) derniere_parution
    -> FROM
    ->   livre
```

```
    -> GROUP BY
    ->   tranche_pages
    -> HAVING -- utilisation des alias
    ->   nb_livres > 1
    -> ORDER BY
    ->   tranche_pages;
+---------------+-----------+-------------------+
| tranche_pages | nb_livres | derniere_parution |
+---------------+-----------+-------------------+
|           300 |         2 |              2008 |
|           500 |         4 |              2016 |
|           700 |         2 |              2016 |
+---------------+-----------+-------------------+
3 rows in set (0.00 sec)

mysql> --
mysql> -- Afficher uniquement les lignes
mysql> -- qui ont au moins deux livres
mysql> -- et une dernière parution postérieure à 2015.
mysql> SELECT
    ->   (nombre_pages DIV 100) * 100 tranche_pages,
    ->   COUNT(*) nb_livres,
    ->   MAX(annee_parution) derniere_parution
    -> FROM
    ->   livre
    -> GROUP BY
    ->   tranche_pages
    -> HAVING -- utilisation des alias
    ->   nb_livres > 1
    ->   AND derniere_parution > 2015
    -> ORDER BY
    ->   tranche_pages;
+---------------+-----------+-------------------+
| tranche_pages | nb_livres | derniere_parution |
+---------------+-----------+-------------------+
|           500 |         4 |              2016 |
|           700 |         2 |              2016 |
+---------------+-----------+-------------------+
2 rows in set (0.00 sec)
```

Pour des raisons de performance, la clause `HAVING` ne doit pas être utilisée pour filtrer des données qui pourraient l'être dans la clause `WHERE`.