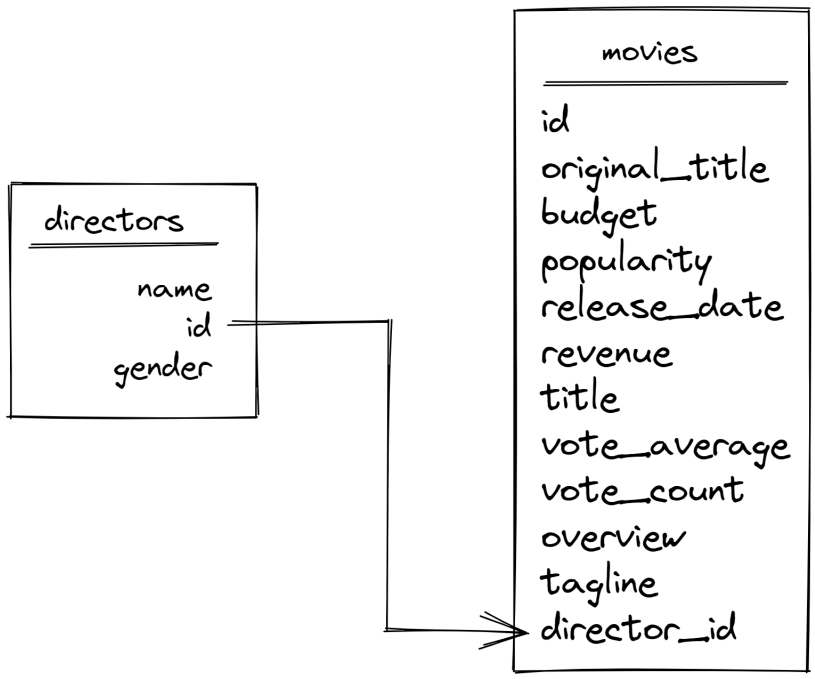
| | | | |
|---|---|---|---|
| **Enseignant :** | Louis RAYNAL | **Établissement :** | ICES |
| **Contact :** | l-raynal@ices.fr | **Année :** | 2025-2026 |
| **Public :** | L3 Maths | | |
| **Ressources :** | https://github.com/LouisRaynal/coursSQL | | |


# Cours-TP 5 : Fonctions d'agrégation et `GROUP BY`

Les données d'une base sont habituellement stockées avec un niveau de détail le plus fin. On recueillera et stockera des informations par entité, plutôt que pour des groupes d'entités. Cependant, pour un but analytique on peut souhaiter agréger des populations d'entités (d'individus). Cette partie du cours s'intéresse donc à comment grouper des données et calculer des informations sur chaque groupe au moyen de fonctions d'agrégation.

**Pratique**

Pour ce cours nous utiliserons la base de données `imdb`. Commencez par télécharger le fichier `imdb.sqlite` sur votre bureau, et connectez-vous-y depuis SQLiteStudio. Le schéma relationnel de cette base de données est rappelé ci-dessous.




## 1. Groupement de lignes avec `GROUP BY`

**Pratique**

Imaginons que vous vouliez compter le nombre de films réalisés par chaque réalisateur dans la base. Après avoir effectué une jointure entre les tables `directors` et `movies`, vous listez simplement tous les noms de réalisateurs. Grâce à la jointure, le nom de chaque réalisateur sera répété autant de fois qu'il y a de lignes dans `movies` lui correspondant.

**Code 1 – Méthode naïve**

```sql
SELECT directors.name
FROM directors
INNER JOIN movies
    ON directors.id = movies.director_id
ORDER BY directors.name
;
```

Vous pourriez ensuite compter à la main combien de fois chaque nom de réalisateur se répète, mais cela prendrait beaucoup trop de temps. C'est là que la clause `GROUP BY` intervient !

Rajoutez la clause `GROUP BY` de la manière suivante :

**Code 2 – Mauvaise utilisation de `GROUP BY`**

```sql
SELECT directors.name
FROM directors
INNER JOIN movies
    ON directors.id = movies.director_id
GROUP BY directors.name
ORDER BY directors.name
;
```

Vous venez de grouper toutes les lignes par valeur de la colonne `name`, c'est à dire par nom de réalisateur. Chaque nom se répète donc une seule fois, et forme un groupe. Mais ici, aucune action n'a été demandée par groupe.

L'action que l'on souhaite est *"compter le nombre de lignes dans chaque groupe"*. Pour ce faire, il faut employer la fonction d'agrégation `COUNT( )` dans la clause `SELECT`.

**Code 3 – Bonne utilisation de `GROUP BY`**

```sql
SELECT directors.name, COUNT(*)
FROM directors
INNER JOIN movies
    ON directors.id = movies.director_id
GROUP BY directors.name
ORDER BY directors.name
;
```

La fonction d'agrégation `COUNT( )` permet de compter le nombre de lignes par groupe, et `*` indique de compter toutes les lignes de la table.

### Attention

La clause `GROUP BY` est toujours utilisée avec une fonction d'agrégation.

Une fois une agrégation faite, nous pouvons filtrer les groupes au moyen du mot clé `HAVING`.

### Pratique

Afin de ne garder que les réalisateurs ayant réalisés plus de 10 films dans la base, nous pouvons adapter la requête précédente de la manière suivante :

Code 4 – Exemple d'utilisation de `HAVING`

```sql
SELECT directors.name, COUNT(*)
FROM directors
INNER JOIN movies
    ON directors.id = movies.director_id
-- WHERE COUNT(*) > 10 ne fonctionnerait pas ici
GROUP BY directors.name
    HAVING COUNT(*) > 10
ORDER BY directors.name
```

Notez que la clause `WHERE` ne peut pas être utilisée pour filtrer le résultat de colonnes agrégées, contrairement à `HAVING`, car ces colonnes n'ont pas encore été calculées au moment de l'évaluation de la clause `WHERE`.

## 2. Fonctions d'agrégation

Les **fonctions d'agrégation** effectuent une opération spécifique en se basant sur les lignes d'un groupe. En SQLite les plus courantes sont les suivantes :

- `COUNT( )` : retourne le nombre de valeurs dans un groupe ;
- `MAX( )` : retourne la valeur maximale dans un groupe ;
- `MIN( )` : retourne la valeur minimale dans un groupe ;
- `AVG( )` : retourne la valeur moyenne dans un groupe ;
- `SUM( )` : retourne la somme des valeurs dans un groupe ;
- `GROUP_CONCAT( , Y)` : retourne la concaténation des valeurs dans un groupe, séparées par la chaîne Y (par exemple `', '`).

### Pratique

Voici un exemple d'utilisation de ces fonctions d'agrégation, pour avoir par réalisateur le nombre films réalisés, la note moyenne maximale, minimale, moyenne, la somme et la concaténation des notes moyennes.

Code 5 – Exemple d'utilisation des fonctions d'agrégation

```sql
SELECT directors.name,
    COUNT(movies.id),
    MAX(movies.vote_average),
    MIN(movies.vote_average),
    AVG(movies.vote_average),
    SUM(movies.vote_average),
    GROUP_CONCAT(movies.vote_average, ', ')
FROM directors
INNER JOIN movies
    ON directors.id = movies.director_id
GROUP BY directors.name
ORDER BY directors.name
```

**Remarque**

Dans la requête précédente, nous avons directement spécifié dans le `COUNT( )` l'identifiant des films au lieu d'utiliser `*`. Dans le premier cas, uniquement les identifiants avec une valeur non `NULL` seront comptées, alors que `*` compte toutes les valeurs, `NULL` ou non. Dans cet exemple cela ne change pas les résultats car il y a autant de lignes que d'identifiants, un par film, mais c'est quelque chose à garder à l'esprit.

Pour vous en assurez, comprenez et lancez les requêtes suivantes.

Code 6 – Exemple d'illustration de `COUNT( )` et valeurs `NULL`

```
CREATE TABLE table_nombres (
    val INTEGER
);

INSERT INTO table_nombres (val) VALUES (1);
INSERT INTO table_nombres (val) VALUES (3);
INSERT INTO table_nombres (val) VALUES (5);
INSERT INTO table_nombres (val) VALUES (NULL);

SELECT COUNT(*) AS NombreLignes, COUNT(val) AS NombreValeurs
FROM table_nombres
;

--DROP TABLE table_nombres;
```

### Comptez les valeurs distinctes

Dans le cas où vous souhaitez compter uniquement les **valeurs distincts** d'une colonnes, au lieu de toutes les valeurs du groupe (distinctes ou non), vous pouvez utiliser le mot clé `DISTINCT` dans la fonction `COUNT( )`.

**Pratique**

La requête suivante compte dans un premier temps tous les identifiants de réalisateurs figurant dans la table `movies`, puis les identifiants **distincts**.

Code 7 – Exemple d'utilisation de `COUNT(DISTINCT ... )`

```
SELECT COUNT(movies.director_id), COUNT(DISTINCT movies.director_id)
FROM movies
;
```

L'ensemble des films a donc été réalisé par 2349 réalisateurs différents.

**Attention**

Remarquez que nous avons utilisé des fonctions d'agrégation sans `GROUP BY`.
**Tant que toutes les expressions après la clause `SELECT` sont des fonctions d'agrégation**, SQL considère qu'il n'y a qu'un groupe : toute la table `movies` dans ce dernier exemple.

### Fonctions d'agrégation appliquées à des expressions

Au lieu d'appeler des fonctions d'agrégation sur des colonnes de la table, vous pouvez très bien les utiliser sur des expressions.

**Pratique**

Voici un exemple où la fonction `MAX( )` est appliquée au produit de deux colonnes, puis à l'arrondie des valeurs d'une colonne :

Code 8 – Exemple d'utilisation des fonctions d'agrégation

```
SELECT directors.name,
    MAX(movies.vote_average * movies.vote_count),
    MAX(ROUND(movies.vote_average))
FROM directors
INNER JOIN movies
    ON directors.id = movies.director_id
GROUP BY directors.name
ORDER BY directors.name
;
```

### Groupement par plusieurs colonnes/expressions

Jusqu'à présent nous n'avons vu que des exemples où un groupement était effectué via les valeurs d'une seule colonne/expression, or nous pouvons **grouper via plusieurs colonnes/expressions**, en spécifiant toutes les expressions selon lesquelles effectuer le groupement après la clause `GROUP BY`.

Lorsque plusieurs colonnes sont spécifiées, le groupement se fait sur les combinaisons de valeurs des colonnes.

**Pratique**

Voici un exemple où l'on groupe par nom de réalisateur, et par note moyenne arrondie, afin de calculer le nombre de films réalisés par un certain réalisateur et ayant reçu une certaine note moyenne arrondie.

Code 9 – Exemple de `GROUP BY` sur une colonne et une expression

```
SELECT directors.name, ROUND(movies.vote_average), COUNT(movies.id)
FROM directors
INNER JOIN movies
    ON directors.id = movies.director_id
GROUP BY directors.name, ROUND(movies.vote_average)
ORDER BY directors.name, 2
;
```

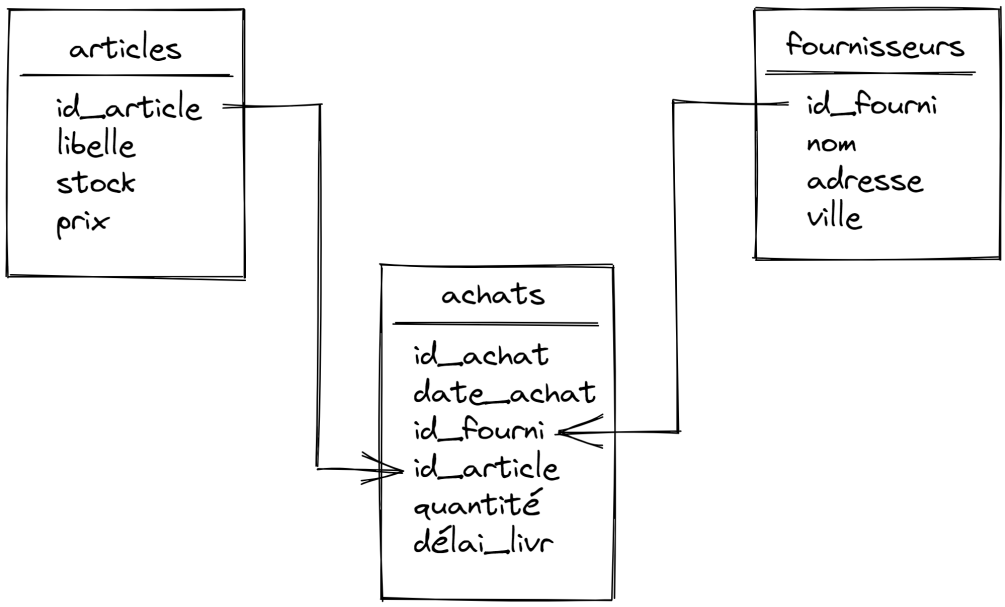
On observe par exemple que le réalisateur Adrian Lyne a réalisé trois films ayant reçu une note moyenne arrondie de 6, et un film avec une note de 7.

**Astuce**

Lorsque vous faites une requête de sélection avec un `GROUP BY`, assurez vous que les colonnes après la clause `SELECT` sans fonction d'agrégation se retrouvent bien dans le `GROUP BY`.

## 3. Exercices

Pour ces exercices vous réutiliserez la base de données `commerce` contenant les données d'achats (table `achats`) de certains articles (table `articles`) vendus par certains fournisseurs (table `fournisseurs`).
Pour rappel, le schéma relationnel de cette base est représenté ci-dessous.




1. Téléchargez sur votre bureau le fichier de base de données `commerce.db`, et connectez-vous-y depuis SQLiteStudio.
2. Calculez le total des valeurs de stock des articles. Donnez un nom de votre choix à ce total.
3. Comptez le nombre de lignes d'achats ainsi que le nombre d'articles distincts achetés (pas la quantité), et auprès de combien de fournisseurs distincts.
4. Affichez pour chaque fournisseur, son identifiant et nom, ainsi que le délai de livraison minimum, maximum et moyen des achats effectués auprès de ces fournisseurs. Affichez même les fournisseurs qui n'ont reçu aucune commande.
5. Modifiez la requête précédente pour n'affichez que les fournisseurs qui ont vendus au moins 2 articles.
6. Affichez pour chaque article, son nombre d'achats différents et la quantité moyenne achetée. Inclure même les articles non achetés s'il y en a.
7. Affichez pour chaque article avec son fournisseur correspondant, le nombre d'achats différents et la quantité moyenne achetée. Affichez même les fournisseurs qui n'ont reçu aucune commande.
8. Affichez le nom des 5 articles ayant rapporté le plus d'argents, la somme rapportée, et affichez la concaténation du nom des fournisseurs en face de chaque article, avec comme délimiteur " - ".