Interrogation d'une base de données SQL

Table des matières

I. Contexte

Durée : 1 h

Environnement de travail : MariaDB

Pré-requis : Savoir faire un `SELECT`, connaître les bases des fonctions

Contexte

Une base de données peut faire bien plus qu'écrire une donnée dans une base puis la restituer. Un SGBD est capable de gérer un lot de données, les filtrer selon des critères bien précis, effectuer des analyses, des sommes, des décomptes, des moyennes...

Ces opérations d'ensemble sont appelées « opérations d'agrégation » et sont indispensables au fonctionnement de nombreux programmes informatiques.

L'objectif de ce cours est d'apprendre à effectuer de telles opérations.

II. Les bases de l'agrégation de données

Objectif

- Apprendre à réaliser des agrégations simples

Mise en situation

Les données unitaires ne sont pas toujours suffisantes pour analyser une situation. Par exemple, un magasin a besoin d'avoir la liste de ses ventes du jour pour sa comptabilité. Mais ce qui va intéresser son gestionnaire, ce n'est pas la liste de toutes les ventes individuelles, mais plutôt le chiffre d'affaires, le total des ventes globales ou le total des ventes par rayon, afin de savoir quels produits ont le plus de succès.

Il ne s'agit plus de trouver une donnée spécifique dans une table, mais de parcourir tout un volume de données et de les analyser selon certains axes précis : faire des sommes, des décomptes, des moyennes...

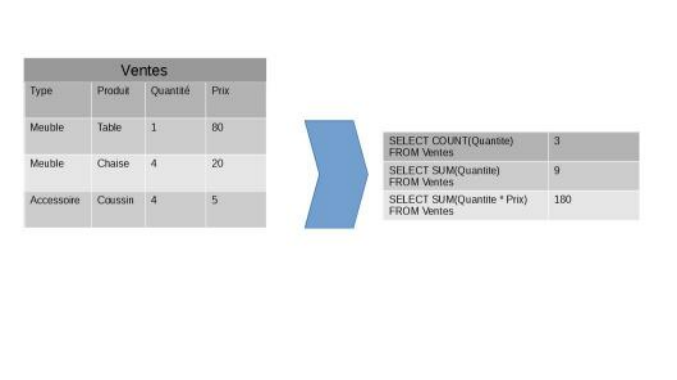
Tout cela est possible grâce à l'**agrégation de données**.

Définition **Fonction d'agrégation**

Une fonction d'agrégation est une fonction qui prend en entrée un ensemble de données et fournit en sortie une valeur unique. Les fonctions les plus courantes en SQL sont :

- `COUNT` : effectue le décompte de l'ensemble des valeurs fournies en entrée
- `SUM` : effectue la somme de l'ensemble des valeurs fournies en entrée
- `MIN` : renvoie la plus petite valeur de l'ensemble
- `MAX` : renvoie la plus grande valeur de l'ensemble
- `GROUP_CONCAT` : concatène l'ensemble des données sous forme d'une chaîne de caractères

Exemple



Voici trois exemples d'agrégation :

- `COUNT` renvoie le nombre de données disponibles. Dans cet exemple, il y a 3 données pour la colonne Quantité (1, 4 et 4).
- `SUM(Quantite)` effectue la somme des valeurs de la colonne Quantité, soit 1 + 4 + 4 = 9

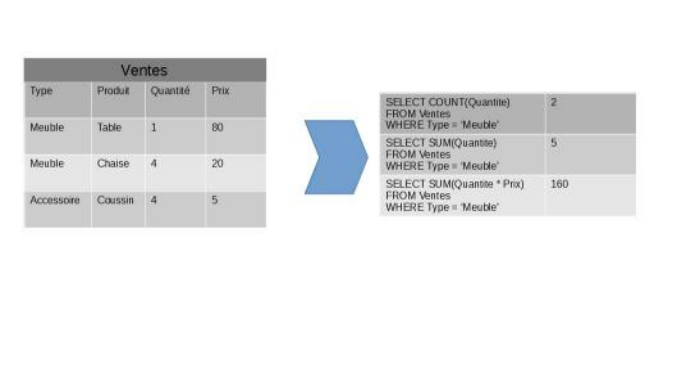
Un ensemble n'est pas limité à une colonne, comme le montre le dernier exemple :

- `SUM(Quantite * Prix)` effectue la somme du produit des valeurs de Quantité et Prix, soit : (1 * 80) + (4 * 20) + (4 * 5) = 180

Agrégation et filtre

Une agrégation n'est pas limitée au contenu de toute une colonne. Grâce à la clause `WHERE` d'une instruction `SELECT`, il est possible de filtrer les données à agréger.

Exemple



Cette fois-ci, les données sont filtrées sur les ventes de meubles. La ligne accessoire est donc exclue.

Syntaxe **À retenir**

- Une fonction d'agrégation transforme un lot de données en une donnée.
- La clause `WHERE` permet de choisir quelles données doivent être incluses dans le calcul d'agrégation.

Complément

Documentation MariaDB[1]

III. Exercice : Appliquez la notion

Question 1

[solution n°1 p.19]

Un magasin dispose d'une table de ventes décrite comme suit :

```
CREATE TABLE Ventes
(
    Id int NOT NULL PRIMARY KEY AUTO_INCREMENT,
    NomVendeur varchar(50) NOT NULL,
    MontantVente decimal(10,2) DEFAULT 0,
    DateVente datetime DEFAULT NOW()
)
;
```

Écrivez une requête permettant de connaître le montant de toutes les ventes du vendeur Dupond.

Question 2

[solution n°2 p.19]

Écrivez une autre requête permettant de connaître le nombre de ventes du vendeur Dupond.

IV. Agrégation par groupe

Objectif

- Apprendre à réaliser des agrégations de groupe

Mise en situation

Une simple agrégation avec filtre suffit rarement pour répondre aux besoins. Par exemple, faire la somme des ventes d'un vendeur permet de connaître le total pour ce vendeur, mais comment obtenir la somme des ventes de chaque vendeur ? S'il y a 10 vendeurs, faut-il effectuer la requête 10 fois ?

Il serait plus pratique de pouvoir demander à la base de données d'effectuer la somme des ventes pour chaque vendeur : ceci est possible, avec la clause `GROUP BY`.

Définition **Clause de groupe**

La clause `GROUP BY` d'une instruction `SELECT` permet de spécifier une liste de colonnes par lesquelles regrouper les données. La base de données va parcourir ces colonnes dans l'ordre, et va regrouper les lignes de la table ayant des valeurs identiques sur ces colonnes.

Une colonne citée dans la clause `GROUP BY` ne sera pas agrégée.

La clause `GROUP BY` doit être écrite après une clause `WHERE`.

1 https://mariadb.com/kb/en/aggregate-functions/

Remarque

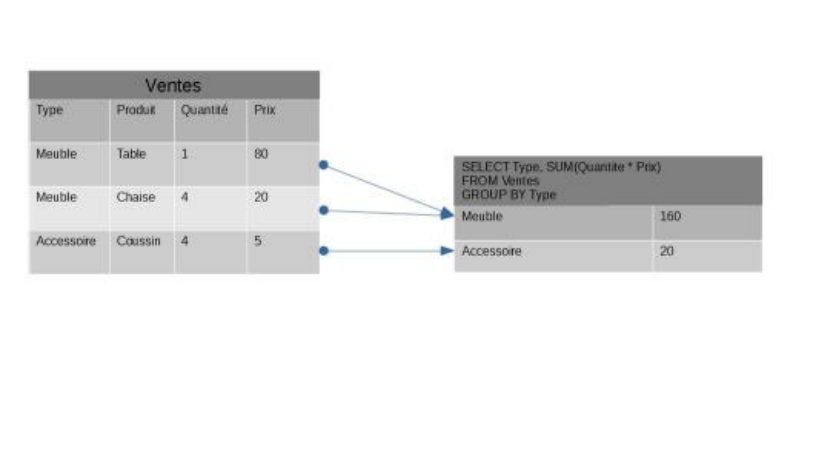
L'ordre des clauses doit respecter un ordre bien précis dans une instruction `SELECT`.

Certaines clauses, comme les `WHERE`, sont facultatives, mais, si elles existent, elles doivent l'être dans cet ordre.

Une instruction `SELECT` avec une clause `GROUP BY` s'écrira donc ainsi :

```
SELECT
    Expressions à renvoyer
FROM
    Table
WHERE
    Conditions
GROUP BY
    Liste des regroupements
```

Exemple



Dans cet exemple, le calcul Quantité * Prix est cette fois-ci regroupé par type de ventes : les lignes ayant le même type sont regroupées ensemble. La colonne Type n'est pas agrégée, seules les colonnes Quantité et Prix le sont. Cela signifie que le résultat final aura une ligne par type différent.

Ce `SELECT` renvoie donc deux valeurs : d'une part, le contenu de la colonne Type, non agrégé, et d'autre, le résultat du calcul Somme du produit des quantités multiplié par le prix unitaire, qui est lui une agrégation.

Fondamental

Il est très important de bien distinguer les colonnes agrégées de celles qui ne le sont pas. Une colonne citée dans une clause `GROUP BY`, et donc non agrégée, peut être utilisée telle quelle dans le `SELECT`. En revanche, une colonne non citée dans une clause `GROUP BY` doit être utilisée dans une fonction d'agrégation.

L'exemple ci-dessus utilise la requête :

```
SELECT
    Type,
    SUM(Quantite * Prix)
FROM Ventes
GROUP BY Type
```

Dans notre cas, la colonne Type peut être utilisée telle quelle dans le `SELECT`, car elle fait partie de la clause `GROUP BY`. Les colonnes Quantité et Prix, en revanche, ne font pas partie de la clause `GROUP BY` : elles ne peuvent donc être intégrées que dans une fonction d'agrégation.

Par exemple, la requête suivante est fausse et renverrait une erreur :

```sql
SELECT
    Type,
    Produit,
    SUM(Quantite * Prix)
FROM Ventes
GROUP BY Type
```

En effet, la colonne Produit n'est ni citée dans la clause `GROUP BY`, ni utilisée au sein d'une fonction d'agrégation.

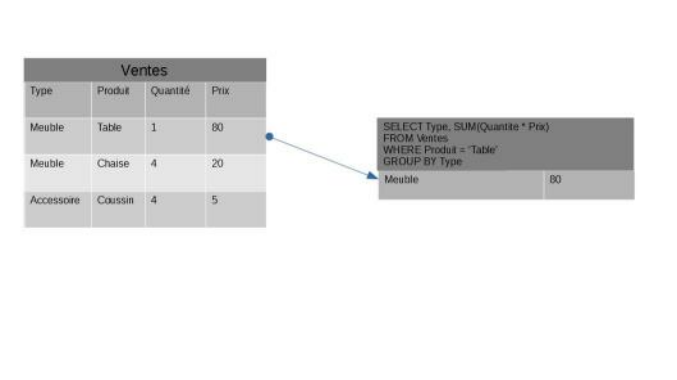
Remarque

Il est bien entendu possible de combiner une clause `WHERE` et une clause `GROUP BY`. Dans ce cas, la clause `GROUP BY` doit être écrite après la clause `WHERE`.

Exemple

La requête ci-dessous combine un `WHERE` et un `GROUP BY` :

```sql
SELECT
    Type,
    SUM(Quantite * Prix)
FROM Ventes
WHERE
    Produit = 'Table'
GROUP BY Type
```



La base de données va d'abord filtrer les lignes répondant à la clause `WHERE`. Dans cet exemple, il ne restera que la première des trois lignes.

Ensuite seulement l'agrégation est effectuée sur les lignes restantes. Ici, il n'y a donc qu'une seule ligne.

Syntaxe **À retenir**

- La clause `GROUP BY` permet de regrouper les données à traiter.
- Les colonnes citées par la clause `GROUP BY` peuvent être utilisées telles quelles dans le `SELECT`. Les autres doivent être intégrées à une fonction d'agrégation.
- La clause `WHERE` effectue un filtre sur les données avant l'opération de regroupement.

Complément

Documentation MariaDB[1]

V. Exercice : Appliquez la notion

Question

[solution n°3 p.19]

Un magasin dispose d'une table de ventes décrite comme suit :

```
CREATE TABLE Ventes
(
    Id int NOT NULL PRIMARY KEY AUTO_INCREMENT,
    NomVendeur varchar(50) NOT NULL,
    MontantVente decimal(10,2) DEFAULT 0,
    DateVente datetime DEFAULT NOW()
)
;
```

Écrivez une requête permettant de connaître le nom des vendeurs et la somme de leurs ventes.

VI. Fenêtrage

Objectif

- Comprendre et maîtriser les fonctions de fenêtrage

Mise en situation

La clause `GROUP BY` vue précédemment permet de regrouper des lignes selon une valeur de colonne et d'effectuer les agrégations selon cette valeur. Cette opération renvoie une ligne par valeur de groupement, et lui affecte le résultat des agrégations demandées pour ce sous-ensemble. Par exemple, il est possible d'obtenir une liste de magasins associée au total des ventes de chaque magasin.

Mais comment faire pour obtenir une liste de magasins et le total des ventes de la région, ou du pays, afin de pouvoir comparer le magasin au total local ?

Sur une région comptant plusieurs magasins, un `GROUP BY` sur la région renverrait une seule ligne, sans le détail des magasins. Un `GROUP BY` sur les magasins ne renverrait pas le total de la région.

Il existe une solution à ce problème : le **fenêtrage**.

Définition **Fenêtrage**

Le fenêtrage signifie que la requête va, pour un calcul précis, effectuer celui-ci selon un groupement déterminé pour ce calcul sans impacter les autres résultats. C'est l'équivalent d'un `GROUP BY`, mais pour lequel le résultat serait dupliqué pour chaque ligne concernée, plutôt que ressemblé en une seule ligne.

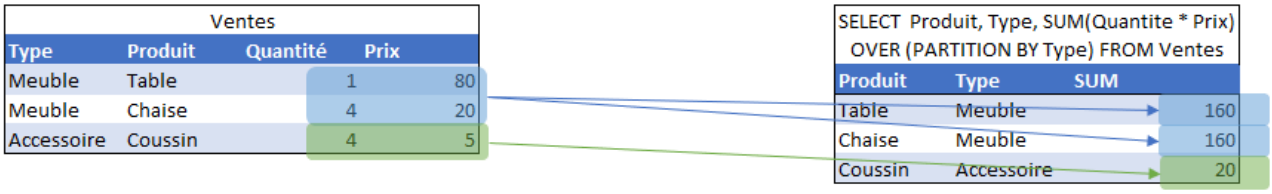
Le fenêtrage s'effectue via le mot-clé `OVER (PARTITION BY ...)`, suivant immédiatement la fonction d'agrégation appelée.

Au contraire d'un groupement, le fenêtrage ne modifie pas le nombre de lignes renvoyées.

1 https://mariadb.com/kb/en/group-by/

Exemple

```sql
SELECT
    Produit, Type,
    SUM(Quantité * Prix) OVER (PARTITION BY Type)
FROM Ventes
```



Ventes

Type	Produit	Quantité	Prix
Meuble	Table	1	80
Meuble	Chaise	4	20
Accessoire	Coussin	4	5

SELECT Produit, Type, SUM(Quantite * Prix) OVER (PARTITION BY Type) FROM Ventes

Produit	Type	SUM
Table	Meuble	160
Chaise	Meuble	160
Coussin	Accessoire	20

Dans cet exemple, le calcul Quantité * Prix est précisé par fenêtrage sur le type. Le `SELECT` renvoie toujours trois lignes, comme dans la table des ventes. Il n'y a pas de regroupement par type. En revanche, la somme effectuée est calculée selon le fenêtrage par type : toutes les ventes du même type auront la même valeur.

Ici, le résultat de 160 correspond à la somme des quantités multipliées par les montants pour tous les meubles (chaises et tables), et ce résultat est repris à chaque ligne de type `Meuble`.

Remarque

Le mot-clé `OVER (PARTITION BY ...)` n'affecte que le calcul effectué précédemment, ici un `SUM`.

Le `PARTITION BY` est facultatif. Dans l'exemple précédent, `SUM(Quantité * Prix) OVER ()` renverrait la valeur 180 sur chaque ligne, soit la somme de l'ensemble des ventes.

Classement et cumul

Le mot-clé `OVER` permet également d'effectuer un calcul par cumul, c'est-à-dire que, pour chaque ligne, on ajoute la valeur de la ligne à celles des lignes précédentes. Cela se fait avec la clause `ORDER BY` au sein du `OVER` : `OVER (ORDER BY ...)`.

Il est possible de cumuler les deux instructions : `OVER (PARTITION BY ... ORDER BY ...)`.

Exemple

Si on veut obtenir le cumul des ventes par type, il faut utiliser la requête :

```sql
SELECT
    Produit, Type,
    SUM(Quantité * Prix) OVER (ORDER BY Produit)
FROM Ventes
```

Ventes

Type	Produit	Quantité	Prix
Meuble	Table	1	80
Meuble	Chaise	4	20
Accessoire	Coussin	4	5

SELECT Produit, Type, SUM(Quantite * Prix) OVER (ORDER BY Produit) FROM Ventes

Produit	Type	SUM
Chaise	Meuble	80
Coussin	Accessoire	100
Table	Meuble	180

Le `SELECT` renvoie toujours trois lignes, comme dans la table des ventes :

- Pour la première ligne, le calcul est effectué sur le produit Quantité * Prix de la première ligne : 4 * 20
- Pour la deuxième ligne, le calcul est effectué cette fois-ci sur la somme des deux premières lignes : 4 * 20 + 4 * 5
- Pour la troisième ligne, le calcul est effectué cette fois-ci sur la somme de toutes les lignes : 4 * 20 + 4 * 5 + 1 * 80

Obtenir un rang

Il est possible de numéroter chaque ligne par ordre d'apparition. Cela se fait via l'instruction `Rank() OVER (ORDER BY ...)`.

Exemple

```sql
SELECT
    Produit,
    Rank() OVER (ORDER BY Produit)
FROM Ventes
```

Ventes

Type	Produit	Quantité	Prix
Meuble	Table	1	80
Meuble	Chaise	4	20
Accessoire	Coussin	4	5

SELECT Produit,
Rank() OVER (ORDER BY Produit)
FROM Ventes

Chaise	1
Coussin	2
Table	3

Le `SELECT` renvoie toujours trois lignes, comme dans la table des ventes. La seconde colonne indique simplement l'ordre des lignes.

Syntaxe **À retenir**

- La clause `OVER(PARTITION BY)` suivant une opération d'agrégation permet d'effectuer un fenêtrage sur celle-ci.
- La clause `OVER(ORDER BY)` permet d'effectuer une opération par cumul.
- La fonction `Rank()` renvoie l'ordre de lignes en fonction d'un ordre spécifié par une clause `OVER(ORDER BY)`.

Complément

Documentation MariaDB[1]

1 https://mariadb.com/kb/en/window-functions-overview/

VII. Exercice : Appliquez la notion

Question

[solution n°4 p.19]

Un magasin dispose d'une table de ventes décrite comme suit :

```sql
CREATE TABLE Ventes
(
    Id INTEGER NOT NULL PRIMARY KEY,
    NomVendeur varchar(50) NOT NULL,
    MontantVente decimal(10,2) DEFAULT 0,
    DateVente datetime DEFAULT NOW()
)
;
```

On souhaite comparer chaque vente au total des ventes du vendeur. Pour cela, écrivez une requête renvoyant la liste des ventes avec les informations suivantes :

- nom du vendeur
- montant de la vente
- montant total des ventes du vendeur

Indice :

La seule agrégation présente ici est celle renvoyant le montant total des ventes du vendeur, et doit s'écrire avec un `SUM(...) OVER (PARTITION BY ...)`.

Indice :

Il n'y a pas besoin de `GROUP BY` pour cette requête, car la seule agrégation se fait sur un `OVER (PARTITION BY)`.

VIII. Filtrer sur une agrégation

Objectif

- Maîtriser les différentes options de filtre sur agrégation

Mise en situation

Les clauses `GROUP BY` et `OVER()` vues précédemment permettent de modifier la façon dont le SGBD agrège les données, mais ces clauses ne modifient pas l'ensemble des données traitées.

Par exemple, un `SELECT` effectué sur une table traitera toutes les lignes de cette table. Les clauses `GROUP BY` ou `OVER()` ne changent que les regroupements ou l'ordre de ces lignes.

Afin de modifier l'ensemble de données retournées, il faut utiliser une clause de filtre. Il existe deux clauses différentes : avant agrégation, et après agrégation.

Définition **Filtre avant agrégation**

Un filtre avant agrégation se fait au moyen de la clause `WHERE` d'une opération `SELECT`.

La clause `WHERE` est suivie d'une ou plusieurs conditions et doit être située après la clause `FROM`, mais avant la clause `GROUP BY`.

Exemple

```sql
SELECT
    Produit,
    SUM(Quantite * Prix)
FROM Ventes
WHERE
    Type = 'Meuble'
GROUP BY
    Produit
```

Fonctionnement de la clause WHERE associée à une agrégation

La clause `WHERE` constitue un filtre avant agrégation. Le SGBD va procéder aux opérations suivantes, dans l'ordre :

1. Récupérer les lignes de données qui valident la condition de la clause `WHERE`
2. Procéder à l'agrégation telle que décrite dans la clause `GROUP BY`
3. Fournir les données calculées à l'utilisateur

Remarque

Il n'est pas possible de placer une condition dans la clause `WHERE` qui porte sur le résultat d'une agrégation, car l'agrégation ne s'est pas encore effectuée à ce stade de la requête.

Par exemple, il n'est pas possible de filtrer de façon à avoir la liste des produits dont les ventes ont dépassé une certaine somme, car cette somme n'est pas encore calculée.

Pour pouvoir effectuer un tel test, il faut utiliser un test après agrégation.

Définition **Filtre après agrégation**

Un filtre après agrégation se fait au moyen de la clause `HAVING` d'une opération `SELECT`.

La clause `HAVING` est suivie d'une ou plusieurs conditions, qui peuvent porter sur un résultat d'agrégation.

Cependant, la formule de calcul doit être précisée à nouveau dans la clause `HAVING`, qui doit être située après la clause `GROUP BY`.

Exemple

```sql
SELECT
    Produit,
    SUM(Quantite * Prix) AS PrixTotal
FROM Ventes
GROUP BY
    Produit
HAVING
    SUM(Quantite * Prix) >= 50
```

L'exemple ci-dessus renvoie la liste des produits dont les ventes cumulées ont dépassé la valeur de 50.

À noter que, même si la somme Quantité * Prix a été renommée « PrixTotal », il n'est pas possible d'écrire `HAVING PrixTotal >= 50`. Il est nécessaire de préciser le calcul `SUM(Quantite * Prix)`.

Fonctionnement de la clause HAVING associée à une agrégation

La clause `HAVING` constitue un filtre après agrégation. Le SGBD va procéder aux opérations suivantes :

1. Procéder à l'agrégation telle que décrite dans la clause `GROUP BY`
2. Ne conserver que les lignes validant la condition de la clause `HAVING`
3. Fournir les données calculées à l'utilisateur

Remarque

Contrairement à une clause `WHERE`, la clause `HAVING` nécessite d'effectuer les calculs sur l'ensemble du lot avant de pouvoir filtrer, puisque la condition du filtre porte sur le résultat de ce calcul.

WHERE et HAVING

La clause `WHERE` et la clause `HAVING` peuvent se combiner. Dans ce cas, le SGBD procède aux opérations dans l'ordre suivant :

1. Filtrer les données selon les conditions décrites dans la clause `WHERE`
2. Procéder à l'agrégation telle que décrite dans la clause `GROUP BY`
3. Ne conserver que les lignes validant la condition de la clause `HAVING`
4. Fournir les données calculées à l'utilisateur

La syntaxe est alors :

```
SELECT
    ...
FROM Table1
WHERE
    Conditions de filtre avant agrégation
GROUP BY
    Liste des groupes
HAVING
    Conditions après agrégation
```

Remarque

L'ordre des clauses est obligatoire. Il n'est pas possible d'écrire un `HAVING` avant un `GROUP BY`, ou un `WHERE` après un `HAVING`.

À noter que l'ordre des clauses correspond à l'ordre des opérations telles qu'effectuées par le SGBD.

Syntaxe **À retenir**

- La clause `WHERE` d'un `SELECT` permet d'effectuer un filtre avant agrégation
- La clause `HAVING` d'un `SELECT` permet d'effectuer un filtre après agrégation
- Une clause `HAVING` doit avoir une clause `GROUP BY`

Complément

Documentation MariaDB[1]

1 https://mariadb.com/kb/en/group-by/

IX. Exercice : Appliquez la notion

Question

[solution n°5 p.19]

Un magasin dispose d'une table de ventes décrite comme suit :

```sql
CREATE TABLE Ventes
(
    Id INTEGER NOT NULL PRIMARY KEY,
    NomVendeur varchar(50) NOT NULL,
    MontantVente decimal(10,2) DEFAULT 0,
    DateVente datetime DEFAULT NOW()
)
;
```

On souhaite offrir une prime à tous les vendeurs dont la somme des ventes depuis l'année dernière, à la même date, dépasse 1000 €. Rédigez une requête permettant de récupérer leur nom et la somme de leurs ventes de cette année.

Indice :

Pour gérer la date, il faut utiliser un filtre avant agrégation : les données des années précédentes ne doivent pas faire partie du calcul.

Indice :

Pour récupérer la date de l'an dernier, il est possible d'utiliser `DATE_ADD`.

Indice :

Le filtre sur le prix supérieur à 1000 € doit être fait après l'agrégation : on veut d'abord calculer la somme, et ensuite filtrer sur le résultat.

X. Limiter les résultats

Objectif

- Apprendre à ne renvoyer qu'un certain nombre de résultats

Mise en situation

Les clauses `WHERE` et `HAVING` permettent d'effectuer des filtres avant et après agrégation, mais cela ne suffit pas toujours. Il est fréquent de ne vouloir récupérer qu'un certain nombre de résultats.

Un classement de performances affichera, par exemple, un top 10. Un bloc de commentaires n'affichera quant à lui que 10 ou 20 commentaires à la fois. On peut également imaginer un système de pagination avec lequel on afficherait les résultats de 21 à 40, puis de 41 à 60...

Cela se fait via une clause de limitation de résultats.

Définition **Limitation des résultats**

La clause `LIMIT` d'une opération `SELECT` permet de limiter les résultats renvoyés par la requête :

- Utilisée avec un seul paramètre numérique, `LIMIT x` permet de ne renvoyer que les x premières lignes de résultats
- Utilisée avec deux paramètres numériques, `LIMIT x, y` signifie d'ignorer les x premières lignes et de renvoyer les y suivantes. Ainsi, les lignes x+1 à x+y sont renvoyées
- `LIMIT x, y` peut également être écrite `LIMIT y OFFSET x`

La clause `LIMIT` doit être écrite après toute autre clause.

Attention

Dans une base de données, si aucun ordre n'est spécifié à l'aide d'une clause `ORDER BY`, il n'y a aucun ordre fiable. Par conséquent, s'il n'y a pas de clause `ORDER BY`, le résultat de `LIMIT` sera aléatoire.

Par exemple, un `SELECT ... LIMIT 1, 10` suivi d'un `SELECT ... LIMIT 11, 10` ne vont pas nécessairement renvoyer les 10 premières lignes, puis les 10 suivantes, car il s'agit de deux requêtes différentes sans ordre précisé.

La seconde instruction pourrait très bien renvoyer des lignes déjà renvoyées dans la première.

Pour utiliser sans risque d'erreur une clause `LIMIT`, il est nécessaire d'avoir une clause `ORDER BY`.

Exemple

```sql
SELECT
    Produit,
    SUM(Quantite * Prix)
FROM Ventes
WHERE
    Type = 'Meuble'
GROUP BY
    Produit
ORDER BY
    SUM(Quantite * Prix) DESC
LIMIT 1
```

Cette opération renvoie au produit le plus vendu en termes de montant de ventes. En effet, la clause `ORDER BY` permet de trier les produits dans l'ordre décroissant des montants de ventes, tandis que la clause `LIMIT` ne renvoie que la première ligne, donc celle avec le plus gros montant de ventes.

Remarque

`LIMIT` est une clause valable dans MariaDB et dans certains SGBD comme PostGreSQL. D'autres SGBD, comme Sql Server ou Oracle, auront une façon différente de parvenir au même résultat. Référez-vous à la documentation du SGBD en question le cas échéant.

Syntaxe **À retenir**

- La clause `LIMIT` d'un `SELECT` permet de sélectionner les lignes à retourner après calcul de la requête.
- `LIMIT x` permet de retourner les x premières lignes de la requête.
- `LIMIT x, y` permet de retourner les lignes de x+1 à x+y.
- Il est nécessaire d'utiliser une clause `ORDER BY` pour obtenir un résultat fiable.

Complément

Documentation MariaDB[1]

1 https://mariadb.com/kb/en/limit/

XI. Exercice : Appliquez la notion

Question

[solution n°6 p.20]

Un magasin dispose d'une table de ventes décrite comme suit :

```
CREATE TABLE Ventes
(
    Id INTEGER NOT NULL PRIMARY KEY,
    NomVendeur varchar(50) NOT NULL,
    MontantVente decimal(10,2) DEFAULT 0,
    DateVente datetime DEFAULT NOW()
)
;
```

Écrivez une requête permettant de renvoyer le nom et le total des ventes des 3 meilleurs vendeurs.

Indice :

Il faut utiliser une clause de tri avant d'utiliser une clause de limite.

XII. Essentiel

XIII. Auto-évaluation

A. Exercice final

Exercice 1

[solution n°7 p.20]

Exercice

Une fonction d'agrégation...

- ○ Est une fonction de calcul renvoyant un résultat à partir d'une valeur
- ○ Est une fonction d'ensemble renvoyant un résultat d'un calcul à partir d'un ensemble de valeurs
- ○ Est une fonction appliquant une transformation sur chaque élément d'un lot de données et renvoie un autre lot de données

Exercice

Pour effectuer la somme d'un lot de valeurs issues d'une même colonne, il faut utiliser...

- ○ L'opérateur +
- ○ La fonction d'agrégation `COUNT`
- ○ La fonction d'agrégation `SUM`

Exercice

Pour effectuer la concaténation d'un lot de valeurs issues d'une même colonne, il faut utiliser...

- ○ La fonction d'agrégation `CONCAT`
- ○ La fonction d'agrégation `GROUP_CONCAT`
- ○ La fonction d'agrégation `COALESCE`

Exercice

La clause `GROUP BY` d'un `SELECT` permet d'effectuer...

- Des agrégations partielles sur chaque élément distinct d'une ou plusieurs colonnes
- Un filtre sur une agrégation
- Un fenêtrage sans réduire le nombre de lignes

Exercice

Les clauses d'un `SELECT` doivent être écrites...

- Sans indication d'ordre précis
- Dans l'ordre `SELECT ... FROM ... GROUP BY ...WHERE ... HAVING ...`
- Dans l'ordre `SELECT ... FROM ... WHERE ... GROUP BY ... HAVING ...`

Exercice

Pour trier les résultats, il faut...

- Utiliser la clause `ORDER BY ...`
- Utiliser la clause `OVER(ORDER BY ...)`
- Une requête `SELECT` ne renvoie pas les données dans un ordre défini

Exercice

Pour effectuer un filtre avant agrégation, il faut employer...

- La clause `WHERE`
- La clause `HAVING`

Exercice

Pour effectuer un filtre après agrégation...

- On peut écrire une agrégation dans la clause `WHERE`
- Il faut donner un alias avec `AS` sur l'agrégation, puis écrire une condition sur cet alias dans une clause `HAVING`
- Il faut écrire une condition sur l'agrégation dans une clause `HAVING`

Exercice

Pour ne renvoyer que les lignes de 21 à 30, il faut écrire...

- LIMIT 21, 30
- LIMIT 20, 10
- LIMIT 21, 10

Exercice

Les différentes clauses `WHERE`, `GROUP BY`, `HAVING`, `ORDER BY` et `LIMIT` sont...

- ○ Identiques dans tous les SGBD
- ○ Différentes dans tous les SGBD
- ○ Certaines sont identiques, mais pas toutes

B. Exercice : Défi

Question 1

[solution n°8 p.22]

Une chaîne de magasins souhaite pouvoir analyser les secteurs géographiques les plus pertinents pour ses futures implémentations. Pour cela, elle souhaiterait un classement des villes dont les magasins génèrent le plus de chiffre d'affaires, accessible depuis une page de son intranet. Pour des raisons d'affichage, au vu du grand nombre de magasins, l'affichage sera limité à 10 magasins par page.

La table des magasins est définie comme suit :

```sql
CREATE TABLE Magasins
(
    Id int NOT NULL PRIMARY KEY AUTO_INCREMENT,
    Nom varchar(50) NOT NULL,
    Ville varchar(100) NOT NULL,
    ChiffreAffaire decimal(10,2) DEFAULT 0
)
;
```

Écrivez la requête `SELECT` permettant de renvoyer la liste des villes, le cumul de tous les chiffres des magasins implémentés dans cette ville et leur rang dans le classement. Cette requête ne devra retourner que les magasins qui devront apparaître sur la première page.

Indice :

Le nom des magasins ne doit pas apparaître dans le résultat, de même que l'identifiant technique.

Indice :

Il n'y a aucun filtre ni avant, ni après agrégation, mais il y a un ordre, qui est nécessaire pour pouvoir obtenir le résultat attendu.

Question 2

[solution n°9 p.22]

Modifiez maintenant la requête pour afficher les éléments de la troisième page.

Indice :

N'oubliez pas la signification exacte du premier paramètre dans une clause `LIMIT x, y`, il y a un piège à cet endroit !

Question 3

[solution n°10 p.23]

La chaîne souhaiterait également comparer le CA de chaque magasin au meilleur magasin de la même ville. Écrivez une requête permettant de retourner le nom, la ville et le chiffre d'affaires de tous les magasins, ainsi que le chiffre d'affaires le plus haut de cette ville. Le résultat devra être trié par ville.

Solutions des exercices

p. 5 Solution n°1

```sql
SELECT
    SUM(MontantVente)
FROM Ventes
WHERE
    NomVendeur = 'Dupond'
```

p. 5 Solution n°2

```sql
SELECT
    COUNT(Id)
FROM Ventes
WHERE
    NomVendeur = 'Dupond'
```

On pourrait également compter le nombre de montants, le résultat serait le même.

p. 8 Solution n°3

```sql
SELECT
    NomVendeur,
    SUM(MontantVente)
FROM
    Ventes
GROUP BY
    NomVendeur
```

p. 11 Solution n°4

```sql
SELECT
    NomVendeur,
    MontantVente,
    SUM(MontantVente) OVER (PARTITION BY NomVendeur)
FROM
    Ventes
```

p. 14 Solution n°5

```sql
SELECT
    NomVendeur,
    SUM(MontantVente)
FROM
    Ventes
WHERE
    DateVente > DATE_ADD(NOW(), INTERVAL -1 YEAR)
GROUP BY
    NomVendeur
HAVING
    SUM(MontantVente) > 1000
```

p. 16 Solution n°6

```sql
SELECT
    NomVendeur,
    SUM(MontantVente)
FROM
    Ventes
GROUP BY
    NomVendeur
ORDER BY
    SUM(MontantVente) DESC
LIMIT 3
```

Exercice p. 16 Solution n°7

Exercice

Une fonction d'agrégation...

- ○ Est une fonction de calcul renvoyant un résultat à partir d'une valeur
- ◉ Est une fonction d'ensemble renvoyant un résultat d'un calcul à partir d'un ensemble de valeurs
- ○ Est une fonction appliquant une transformation sur chaque élément d'un lot de données et renvoie un autre lot de données

Une fonction d'agrégation reçoit en entrée un ensemble de valeurs, mais ne renvoie qu'une valeur unique. Par exemple, une somme, un décompte...

Exercice

Pour effectuer la somme d'un lot de valeurs issues d'une même colonne, il faut utiliser...

- ○ L'opérateur `+`
- ○ La fonction d'agrégation `COUNT`
- ◉ La fonction d'agrégation `SUM`

La fonction d'agrégation `SUM` permet d'effectuer une somme. `COUNT` est un décompte. Quant à l'opérateur `+`, il permet d'effectuer la somme entre deux valeurs discrètes, et non pas un lot de valeurs d'une même colonne.

Exercice

Pour effectuer la concaténation d'un lot de valeurs issues d'une même colonne, il faut utiliser...

- ○ La fonction d'agrégation `CONCAT`
- ◉ La fonction d'agrégation `GROUP_CONCAT`
- ○ La fonction d'agrégation `COALESCE`

Il s'agit de la fonction d'agrégation `GROUP_CONCAT`. `CONCAT` n'est pas une fonction d'agrégation, pas plus que `COALESCE`.

Exercice

La clause `GROUP BY` d'un `SELECT` permet d'effectuer...

- [x] Des agrégations partielles sur chaque élément distinct d'une ou plusieurs colonnes
- [ ] Un filtre sur une agrégation
- [ ] Un fenêtrage sans réduire le nombre de lignes

Ne pas confondre une clause `GROUP BY` de la syntaxe `OVER(PARTITION BY)`, qui ne s'applique qu'à une seule expression.

Exercice

Les clauses d'un `SELECT` doivent être écrites...

- [ ] Sans indication d'ordre précis
- [ ] Dans l'ordre `SELECT ... FROM ... GROUP BY ...WHERE ... HAVING ...`
- [x] Dans l'ordre `SELECT ... FROM ... WHERE ... GROUP BY ... HAVING ...`

Les clauses sont dans l'ordre d'exécution. Le `WHERE` étant une clause de filtre avant agrégation, elle est écrite avant la clause `GROUP BY`.

Exercice

Pour trier les résultats, il faut...

- [x] Utiliser la clause `ORDER BY ...`
- [ ] Utiliser la clause `OVER(ORDER BY ...)`
- [ ] Une requête `SELECT` ne renvoie pas les données dans un ordre défini

Ne pas confondre la clause `ORDER BY`, qui permet de trier les résultats et l'instruction `OVER(ORDER BY)`, qui permet d'effectuer une agrégation cumulée.

De plus, une requête `SELECT` n'a pas d'ordre défini seulement si aucun ordre n'est précisé. Il est justement possible de spécifier un ordre via la clause `ORDER BY`.

Exercice

Pour effectuer un filtre avant agrégation, il faut employer...

- [x] La clause `WHERE`
- [ ] La clause `HAVING`

`WHERE` permet de faire un filtre avant agrégation, `HAVING` un filtre après agrégation.

Exercice

Pour effectuer un filtre après agrégation...

- [ ] On peut écrire une agrégation dans la clause `WHERE`
- [ ] Il faut donner un alias avec `AS` sur l'agrégation, puis écrire une condition sur cet alias dans une clause `HAVING`
- [x] Il faut écrire une condition sur l'agrégation dans une clause `HAVING`

Même si un calcul d'agrégation est déjà effectué dans le bloc `SELECT`, il faut le réécrire dans le bloc `HAVING`.

Exercice

Pour ne renvoyer que les lignes de 21 à 30, il faut écrire...

- ○ LIMIT 21, 30
- ◉ LIMIT 20, 10
- ○ LIMIT 21, 10

Q `LIMIT x, y` signifie qu'il faut ignorer les x premières lignes et ne prendre que les y suivantes. Par conséquent, pour lire les lignes 21 à 30, il faut donc ignorer les lignes 1 à 20 et prendre les 10 suivantes.

Exercice

Les différentes clauses `WHERE`, `GROUP BY`, `HAVING`, `ORDER BY` et `LIMIT` sont...

- ○ Identiques dans tous les SGBD
- ○ Différentes dans tous les SGBD
- ◉ Certaines sont identiques, mais pas toutes

Q Si un certain nombre de syntaxes sont respectées par les différents SGBD du marché, ce n'est malheureusement pas le cas de toutes.

Les clauses `WHERE`, `GROUP BY`, `HAVING` et `ORDER BY` sont identiques pour tous les SGBD majeurs. Par contre, certaines clauses comme `LIMIT` n'existent pas dans tous les SGBD.

Il est nécessaire de se référer à la documentation du SGBD utilisé en cas de doute.

p. 18 **Solution n°8**

```sql
SELECT
    Ville,
    SUM(ChiffreAffaire),
    Rank() OVER (ORDER BY SUM(ChiffreAffaire) DESC)
FROM
    Magasins
GROUP BY
    Ville
ORDER BY
    SUM(ChiffreAffaire) DESC
LIMIT 10
```

p. 18 **Solution n°9**

```sql
SELECT
    Ville,
    SUM(ChiffreAffaire),
    Rank() OVER (ORDER BY SUM(ChiffreAffaire) DESC)
FROM
    Magasins
GROUP BY
    Ville
ORDER BY
    SUM(ChiffreAffaire) DESC
LIMIT 20, 10
```

p. 18 **Solution n°10**

```sql
SELECT
    Nom, Ville, ChiffreAffaire,
    MAX(ChiffreAffaire) OVER (PARTITION BY Ville)
FROM
    Magasins
ORDER BY
   Ville
```